

ARCOSS

LNCS 8889

Hee-Kap Ahn  
Chan-Su Shin (Eds.)

# Algorithms and Computation

25th International Symposium, ISAAC 2014  
Jeonju, Korea, December 15–17, 2014  
Proceedings

 Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison, UK

Josef Kittler, UK

John C. Mitchell, USA

Bernhard Steffen, Germany

Demetri Terzopoulos, USA

Gerhard Weikum, Germany

Takeo Kanade, USA

Jon M. Kleinberg, USA

Friedemann Mattern, Switzerland

Moni Naor, Israel

C. Pandu Rangan, India

Doug Tygar, USA

## Advanced Research in Computing and Software Science

Subline of Lecture Notes in Computer Science

### Subline Series Editors

Giorgio Ausiello, *University of Rome 'La Sapienza', Italy*

Vladimiro Sassone, *University of Southampton, UK*

### Subline Advisory Board

Susanne Albers, *TU Munich, Germany*

Benjamin C. Pierce, *University of Pennsylvania, USA*

Bernhard Steffen, *University of Dortmund, Germany*

Deng Xiaotie, *City University of Hong Kong*

Jeannette M. Wing, *Microsoft Research, Redmond, WA, USA*

More information about this series at <http://www.springer.com/series/7407>

Hee-Kap Ahn · Chan-Su Shin (Eds.)

# Algorithms and Computation

25th International Symposium, ISAAC 2014  
Jeonju, Korea, December 15–17, 2014  
Proceedings

*Editors*

Hee-Kap Ahn  
Pohang University of Science and Technology  
Pohang  
Korea, Republic of (South Korea)

Chan-Su Shin  
Hankuk University of Foreign Studies  
Yongin-si  
Korea, Republic of (South Korea)

ISSN 0302-9743  
ISBN 978-3-319-13074-3  
DOI 10.1007/978-3-319-13075-0

ISSN 1611-3349 (electronic)  
ISBN 978-3-319-13075-0 (eBook)

Library of Congress Control Number: 2014955201

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Springer Cham Heidelberg New York Dordrecht London

© Springer International Publishing Switzerland 2014

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Preface

The papers in this volume were presented at the 25th International Symposium on Algorithms and Computation (ISAAC 2014), held in Jeonju, South Korea, during December 15–17, 2014. In the past, ISAAC was held in Tokyo (1990), Taipei (1991), Nagoya (1992), Hong Kong (1993), Beijing (1994), Cairns (1995), Osaka (1996), Singapore (1997), Taejon (1998), Chennai (1999), Taipei (2000), Christchurch (2001), Vancouver (2002), Kyoto (2003), Hong Kong (2004), Hainan (2005), Kolkata (2006), Sendai (2007), Gold Coast (2008), Hawaii (2009), Jeju (2010), Yokohama (2011), Taipei (2012), and Hong Kong (2013) over 25 years from 1990 to 2014.

ISAAC is an acclaimed annual international symposium that covers a wide range of topics in algorithms and theory of computation, and that provides a forum for researchers where they can exchange ideas in this active research community. In response to the call for papers, ISAAC 2014 received 171 submissions from 38 countries. Each submission was reviewed by at least three Program Committee members with the assistance of 189 external reviewers. Through extensive discussion, the Program Committee selected 60 papers for presentation in ISAAC 2014. Two special issues, one of *Algorithmica* and one of *International Journal of Computational Geometry and Applications*, are prepared for some selected papers among the presented ones in ISAAC 2014.

The best paper award was given to “Concentrated Hitting Times of Randomized Search Heuristics with Variable Drift” by Per Kristian Lehre and Carsten Witt. Two eminent invited speakers, Ulrik Brandes from University of Konstanz, Germany and Giuseppe F. Italiano from Università di Roma “Tor Vergata”, Italy, gave interesting invited talks at the conference.

We would like to thank all Program Committee members and external reviewers for their excellent work in the difficult review and selection process. We would like to thank all authors who submitted papers for our consideration; they all contributed to the high quality of the conference. We would like to thank Conference Chair Kunsoo Park and Organizing Committee members for their dedicated contribution. Finally, we would like to thank our conference volunteers, sponsor SRC-GAIA (Center for Geometry and Its Applications), and supporting organizations KIISE (The Korean Institute of Information Scientists and Engineers) and SIGTCS (Special Interest Group on Theoretical Computer Science) of KIISE for their assistance and support.

December 2014

Hee-Kap Ahn  
Chan-Su Shin

# Organization

## Program Committee

Hee-Kap Ahn	Pohang University of Science and Technology, South Korea
Peter Brass	City College of New York, USA
Gerth Stølting Brodal	Aarhus University, Denmark
Xavier Goaoc	University Paris-Est Marne-la-Vallée, France
Simon Gog	University of Melbourne, Australia
Mordecai Golin	Hong Kong University of Science and Technology, Hong Kong
Roberto Grossi	University of Pisa, Italy
Sungjin Im	University of California, Merced, USA
Rahul Jain	National University of Singapore, Singapore
Akinori Kawachi	Tokyo Institute of Technology, Japan
Christian Knauer	Universität Bayreuth, Germany
Pinyan Lu	Microsoft Research Asia, China
Kazuhisa Makino	RIMS, Kyoto University, Japan
Peter Bro Miltersen	Aarhus University, Denmark
Wolfgang Mulzer	Freie Universität Berlin, Germany
Joong Chae Na	Sejong University, South Korea
Srinivasa Rao Satti	Seoul National University, South Korea
Saket Saurabh	Institute of Mathematical Sciences, India
Tetsuo Shibuya	University of Tokyo, Japan
Chan-Su Shin	Hankuk University of Foreign Studies, South Korea
Michiel Smid	Carleton University, Canada
Hisao Tamaki	Meiji University, Japan
Gerhard Woeginger	Eindhoven University of Technology, The Netherlands
Alexander Wolff	Universität Würzburg, Germany
Bang Ye Wu	National Chung Cheng University, Taiwan
Chee Yap	New York University, USA
Hsu-Chun Yen	National Taiwan University, Taiwan
Louxin Zhang	National University of Singapore, Singapore
Peng Zhang	Shandong University, China
Xiao Zhou	Tohoku University, Japan
Binhai Zhu	Montana State University, USA

## Additional Reviewers

Alt, Helmut  
Anagnostopoulos, Aris  
Anshu, Anurag  
Antoniadis, Antonios  
Asinowski, Andrei  
Bae, Sang Won  
Barba, Luis  
Barbay, J  r  my  
Bille, Philip  
Bonichon, Nicolas  
Bonsma, Paul  
Brandstadt, Andreas  
Braverman, Vladimir  
Cabello, Sergio  
Cela, Eranda  
Chang, Ching-Lueh  
Chang, Jou-Ming  
Chen, Ho-Lin  
Chen, Jiecao  
Chen, Xin  
Chlamtac, Eden  
Colin de Verdi  re,   ric  
Da Lozzo, Giordano  
Devillers, Olivier  
Dobbins, Michael Gene  
D  rr, Christoph  
Elbassioni, Khaled  
Elmasry, Amr  
Epstein, Leah  
Fernau, Henning  
Fiorini, Samuel  
Fleszar, Krzysztof  
Fuchs, Fabian  
Fukunaga, Takuro  
Giannopoulos, Panos  
Giaquinta, Emanuele  
Golovach, Petr  
Grunert, Romain  
Gunawan, Andreas D.M.  
Gupta, Ankur  
Gupta, Sushmita  
Gurski, Frank  
Hajiaghayi, Mohammadtaghi  
Hatano, Kohei

He, Meng  
Henze, Matthias  
Higashikawa, Yuya  
Hsieh, Sun-Yuan  
Huang, Guan-Shieng  
Hubard, Alfredo  
Imai, Tatsuya  
Ishii, Toshimasa  
Ito, Takehiro  
Ivanyos, Gabor  
Jaume, Rafel  
Jiang, Minghui  
Jo, Seungbum  
Johnson, Matthew  
Kakoulis, Konstantinos  
Kamiyama, Naoyuki  
Kavitha, Telikepalli  
Kim, Heuna  
Kim, Jin Wook  
Kim, Sung-Ryul  
Kindermann, Philipp  
Kiraly, Tamas  
Klauck, Hartmut  
Kobayashi, Yusuke  
Kolay, Sudeshna  
Kortsarz, Guy  
Kratsch, Dieter  
Kriegel, Klaus  
Kuang, Jian  
Kulkarni, Raghav  
Laekhanukit, Bundit  
Lampis, Michael  
Langetepe, Elmar  
Le Gall, Francois  
Lee, Inbok  
Lee, Mun-Kyu  
Lee, Troy  
Leike, Jan  
Levin, Asaf  
Li, Liang  
Liao, Chung-Shou  
Lin, Chengyu  
Lin, Chun-Cheng  
Liotta, Giuseppe



Liu, Jingcheng  
Liu, Zhengyang  
M.S., Ramanujan  
Mcauley, Julian  
Megow, Nicole  
Mestre, Julian  
Misra, Neeldhara  
Miura, Kazuyuki  
Mizuki, Takaaki  
Mondal, Debajyoti  
Montanaro, Ashley  
Montenegro, Ravi  
Mori, Ryuhei  
Mukherjee, Joydeep  
Mustafa, Nabil  
Navarro, Gonzalo  
Nies, Andre  
Nishimura, Harumichi  
O Dunlaing, Colm  
Ochem, Pascal  
Ohlebusch, Enno  
Okamoto, Yoshio  
Onodera, Taku  
Osipov, Vitaly  
Otachi, Yota  
Oudot, Steve  
Panolan, Fahad  
Park, Heejin  
Paulusma, Daniel  
Peleg, David  
Peng, Dongliang  
Peng, Pan  
Petri, Matthias  
Pilaud, Vincent  
Pilipczuk, Michal  
Poon, Sheung-Hung  
Praveen, M.  
Pruhs, Kirk  
Pérez-Lantero, Pablo  
Rahman, Md. Saidur  
Rai, Ashutosh  
Rautenbach, Dieter  
Rotbart, Noy  
Rote, Günter  
Rutter, Ignaz  
Sabharwal, Yogish  
Sadakane, Kunihiko  
Sarrabezolles, Pauline  
Schneider, Stefan  
Schulz, André  
Seiferth, Paul  
Seto, Kazuhisa  
Shah, Rahul  
Shao, Mingfu  
Shioura, Akiyoshi  
Sim, Jeong Seop  
Sitters, Rene  
Spoerhase, Joachim  
Stehn, Fabian  
Stein, Yannik  
Suzuki, Akira  
Sæther, Sigve Hortemo  
Takazawa, Kenjiro  
Tamaki, Suguru  
Tani, Seiichiro  
Tanigawa, Shin-Ichi  
Ting, Chuan-Kang  
Tong, Weitian  
Tsur, Dekel  
Uchizawa, Kei  
Uno, Takeaki  
Upadhyay, Sarvagya  
van Iersel, Leo  
van Stee, Rob  
Vialeite, Stephane  
Vind, Søren  
Wahlström, Magnus  
Wakabayashi, Yoshiko  
Wang, Bow-Yaw  
Wang, Hung-Lung  
Wang, Menghui  
Wang, Yue-Li  
Wu, Chenchen  
Wu, Zhilin  
Wulff-Nilsen, Christian  
Xia, Mingji  
Xiao, Mingyu  
Xiao, Tao  
Xu, Ning  
Yamamoto, Masaki  
Yang, Chang-Biau  
Yang, De-Nian

X Organization

Yang, Kuan  
Yasunaga, Kenji  
Ye, Deshi  
Zaffanella, Enea  
Zhang, Chihao

Zhang, Jialin  
Zhang, Shengyu  
Zhang, Yong  
Zielinski, Pawel

# Invited Talks

# Biconnectivity in Directed Graphs<sup>\*</sup>

Giuseppe F. Italiano

Univ. of Rome “Tor Vergata”, Via del Politecnico 1, 00133 Roma, Italy

`giuseppe.italiano@uniroma2.it`

Edge and vertex connectivity are fundamental concepts in graph theory with numerous practical applications. Given an undirected graph  $G = (V, E)$ , an edge is a *bridge* if its removal increases the number of connected components of  $G$ . Graph  $G$  is 2-edge-connected if it has no bridges. The 2-edge-connected components of  $G$  are its maximal 2-edge-connected subgraphs. Two vertices  $v$  and  $w$  are 2-edge-connected if there are two edge-disjoint paths between  $v$  and  $w$ : we denote this relation by  $v \leftrightarrow_{2e} w$ . Equivalently, by Menger’s Theorem,  $v$  and  $w$  are 2-edge-connected if the removal of any edge leaves them in the same connected component. Analogous definitions can be given for 2-vertex connectivity. In particular, a vertex is an *articulation point* if its removal increases the number of connected components of  $G$ . A graph  $G$  is 2-vertex-connected if it has at least three vertices and no articulation points. The 2-vertex-connected components of  $G$  are its maximal 2-vertex-connected subgraphs. Two vertices  $v$  and  $w$  are 2-vertex-connected if there are two internally vertex-disjoint paths between  $v$  and  $w$ : we denote this relation by  $v \leftrightarrow_{2v} w$ . If  $v$  and  $w$  are 2-vertex-connected then Menger’s Theorem implies that the removal of any vertex different from  $v$  and  $w$  leaves them in the same connected component. The converse does not necessarily hold, since  $v$  and  $w$  may be adjacent but not 2-vertex-connected. It is easy to show that  $v \leftrightarrow_{2e} w$  (resp.,  $v \leftrightarrow_{2v} w$ ) if and only if  $v$  and  $w$  are in a same 2-edge-connected (resp., 2-vertex-connected) component. All bridges, articulation points, 2-edge- and 2-vertex-connected components of undirected graphs can be computed in linear time essentially by the same algorithm based on depth-first search.

While edge and vertex connectivity have been thoroughly studied in the case of undirected graphs, surprisingly not much has been investigated for directed graphs. Given a directed graph  $G$ , an edge (resp., a vertex) is a *strong bridge* (resp., a *strong articulation point*) if its removal increases the number of strongly connected components of  $G$ . A directed graph  $G$  is 2-edge-connected (resp., 2-vertex-connected) if it has no strong bridges (resp., strong articulation points and has at least three vertices). The 2-edge-connected (resp., 2-vertex-connected) components of  $G$  are its maximal 2-edge-connected (resp., 2-vertex-connected) subgraphs. Similarly to the undirected case, we say that two vertices  $v$  and  $w$  are 2-edge-connected (resp., 2-vertex-connected), and we denote this relation by

---

<sup>\*</sup> Work partially supported by the Italian Ministry of Education, University and Research, under Project AMANDA (Algorithmics for MAssive and Networked DATA).

$v \leftrightarrow_{2e} w$  (resp.,  $v \leftrightarrow_{2v} w$ ), if there are two edge-disjoint (resp., internally vertex-disjoint) directed paths from  $v$  to  $w$  and two edge-disjoint (resp., internally vertex-disjoint) directed paths from  $w$  to  $v$ . (Note that a path from  $v$  to  $w$  and a path from  $w$  to  $v$  need not be edge-disjoint or vertex-disjoint). It is easy to see that  $v \leftrightarrow_{2e} w$  if and only if the removal of any edge leaves  $v$  and  $w$  in the same strongly connected component. Similarly,  $v \leftrightarrow_{2v} w$  implies that the removal of any vertex different from  $v$  and  $w$  leaves  $v$  and  $w$  in the same strongly connected component. We define a *2-edge-connected block* (resp., *2-vertex-connected block*) of a directed graph  $G = (V, E)$  as a maximal subset  $B \subseteq V$  such that  $u \leftrightarrow_{2e} v$  (resp.,  $u \leftrightarrow_{2v} v$ ) for all  $u, v \in B$ . It can be seen that, differently from undirected graphs, in directed graphs 2-edge- and 2-vertex-connected blocks do not correspond to 2-edge-connected and 2-vertex-connected components.

Furthermore, these notions seem to have a much richer (and more complicated) structure in directed graphs. Just to give an example, we observe that while in the case of undirected connected graphs the 2-edge-connected components (which correspond to the 2-edge-connected blocks) are exactly the connected components left after the removal of all bridges, for directed strongly connected graphs the 2-edge-connected components, the 2-edge-connected blocks, and the strongly connected components left after the removal of all strong bridges are not necessarily the same.

In this talk, we survey some very recent work on 2-edge and 2-vertex connectivity in directed graphs, both from the theoretical and the practical viewpoint.

# Social Network Algorithmics<sup>\*</sup>

Ulrik Brandes

Computer & Information Science, University of Konstanz

Network science is a burgeoning domain of data analysis in which the focus is on structures and dependencies rather than populations and independence [1]. Social network analysis is network science applied to the empirical study of social structures, typically utilizing observations on social relationships to analyze the actors involved in them [2].

Methods for the analysis of social networks abound. They include, for instance, numerous centrality indices, vertex equivalences, and clustering techniques, many of which are applied on networks in other disciplines as well. For substantively oriented analysts, however, it is often difficult to choose, let alone justify, a particular variant method. Similarly, it is difficult for researchers interested in computational aspects to understand which methods are worthwhile to consider and whether variants and restrictions are meaningful and relevant.

In an attempt to bridge the gap between theory and methods, and drawing on a substantial record of interdisciplinary cooperation, we have developed a comprehensive research program, *the positional approach to network analysis*. It provides a unifying framework for network analysis in the pursuit of two closely related goals:

1. to establish a *science* of networks, and
2. to facilitate mathematical and algorithmic research.

The first caters to methodologists and social scientists: by embracing measurement theory, network-analytic methods are opened up for theoretical justification and detailed empirical testing. The second caters to mathematicians and computer scientists: by structuring the space of methods, gaps and opportunities are exposed.

After a brief introduction and delineation of network science and social network analysis, the main elements of the positional approach are introduced in this talk. I will then concentrate on exemplary instantiations for analytic concepts such as centrality, roles, and cohesion. Particular emphasis is placed on resulting combinatorial and algorithmic challenges involving, for instance, partial orders, graphs, and path algebras.

---

<sup>\*</sup> I gratefully acknowledge financial support from DFG under grant Br 2158/6-1.

## References

1. Brandes, U., Robins, G., McCranie, A., Wasserman, S.: What is network science? *Network Science* 1(1), 1–15 (2013)
2. Hennig, M., Brandes, U., Pfeffer, J., Mergel, I.: *Studying Social Networks – A Guide to Empirical Research*. Campus, Frankfurt/New York (2012)

# Contents

## Computational Geometry I

- Line-Constrained  $k$ -Median,  $k$ -Means, and  $k$ -Center Problems in the Plane . . . 3  
*Haitao Wang and Jingru Zhang*
- Reconstructing Point Set Order Types from Radial Orderings . . . . . 15  
*Oswin Aichholzer, Jean Cardinal, Vincent Kusters, Stefan Langerman,  
and Pavel Valtr*
- A Randomized Divide and Conquer Algorithm for Higher-Order Abstract  
Voronoi Diagrams. . . . . 27  
*Cecilia Bohler, Chih-Hung Liu, Evanthia Papadopoulou,  
and Maksym Zavershynskiy*

## Combinatorial Optimization I

- Average-Case Complexity of the Min-Sum Matrix Product Problem. . . . . 41  
*Ken Fong, Minming Li, Hongyu Liang, Linji Yang, and Hao Yuan*
- Efficiently Correcting Matrix Products . . . . . 53  
*Leszek Gąsieniec, Christos Levcopoulos, and Andrzej Lingas*
- 3D Rectangulations and Geometric Matrix Multiplication . . . . . 65  
*Peter Floderus, Jesper Jansson, Christos Levcopoulos,  
Andrzej Lingas, and Dzmitry Sledneu*

## Graph Algorithms: Enumeration

- Enumeration of Maximum Common Subtree Isomorphisms  
with Polynomial-Delay . . . . . 81  
*Andre Droschinsky, Bernhard Heinemann, Nils Kriege, and Petra Mutzel*
- Efficient Enumeration of Induced Subtrees in a  $K$ -Degenerate Graph . . . . . 94  
*Kunihiro Wasa, Hiroki Arimura, and Takeaki Uno*
- An Efficient Method for Indexing All Topological Orders  
of a Directed Graph . . . . . 103  
*Yuma Inoue and Shin-ichi Minato*



**Matching and Assignment I**

Planar Matchings for Weighted Straight Skeletons . . . . .	117
<i>Therese Biedl, Stefan Huber, and Peter Palfrader</i>	
Orienting Dynamic Graphs, with Applications to Maximal Matchings and Adjacency Queries . . . . .	128
<i>Meng He, Ganggui Tang, and Norbert Zeh</i>	
Dynamic and Multi-Functional Labeling Schemes . . . . .	141
<i>Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Noy Rotbart</i>	

**Data Structures and Algorithms I**

Hashing and Indexing: Succinct Data Structures and Smoothed Analysis. . . . .	157
<i>Alberto Policriti and Nicola Prezza</i>	
Top- $k$ Term-Proximity in Succinct Space. . . . .	169
<i>J. Ian Munro, Gonzalo Navarro, Jesper Sindahl Nielsen, Rahul Shah, and Sharma V. Thankachan</i>	
The Power and Limitations of Static Binary Search Trees with Lazy Finger. . . . .	181
<i>Presejit Bose, Karim Douïeb, John Iacono, and Stefan Langerman</i>	

**Fixed-Parameter Tractable Algorithms**

Minimum-Cost $b$ -Edge Dominating Sets on Trees . . . . .	195
<i>Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto</i>	
Fixed-Parameter Tractability of Token Jumping on Planar Graphs . . . . .	208
<i>Takehiro Ito, Marcin Kamiński, and Hirotaka Ono</i>	
Covering Problems for Partial Words and for Indeterminate Strings . . . . .	220
<i>Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń</i>	

**Scheduling Algorithms**

Dynamic Interval Scheduling for Multiple Machines. . . . .	235
<i>Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu</i>	
Throughput Maximization in Multiprocessor Speed-Scaling. . . . .	247
<i>Eric Angel, Evripidis Bampis, Vincent Chau, and Nguyen Kim Thang</i>	
Speed-Scaling with No Preemptions . . . . .	259
<i>Evripidis Bampis, Dimitrios Letsios, and Giorgio Lucarelli</i>	

**Computational Complexity**

A Short Implicant of a CNF Formula with Many Satisfying Assignments . . . 273  
*Daniel M. Kane and Osamu Watanabe*

On the Computational Complexity of Vertex Integrity and Component  
 Order Connectivity . . . . . 285  
*Pål Grønås Drange, Markus Sortland Dregi, and Pim van't Hof*

Co-Clustering Under the Maximum Norm . . . . . 298  
*Laurent Bulteau, Vincent Froese, Sepp Hartung, and Rolf Niedermeier*

**Computational Geometry II**

The Price of Order . . . . . 313  
*Prosenjit Bose, Pat Morin, and André van Renssen*

Range Queries on Uncertain Data . . . . . 326  
*Jian Li and Haitao Wang*

On the Most Likely Voronoi Diagram and Nearest Neighbor Searching . . . . 338  
*Subhash Suri and Kevin Verbeek*

**Approximation Algorithms**

An Improved Approximation Algorithm for the Minimum Common  
 Integer Partition Problem . . . . . 353  
*Weitian Tong and Guohui Lin*

Positive Semidefinite Relaxation and Approximation Algorithm for Triple  
 Patterning Lithography . . . . . 365  
*Tomomi Matsui, Yukihide Kohira, Chikaaki Kodama, and Atsushi Takahashi*

An FPTAS for the Volume Computation of 0-1 Knapsack Polytopes Based  
 on Approximate Convolution Integral . . . . . 376  
*Ei Ando and Shuji Kijima*

**Graph Theory and Algorithms**

Polynomial-Time Algorithm for Sliding Tokens on Trees . . . . . 389  
*Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang,  
 Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada*

Minimal Obstructions for Partial Representations of Interval Graphs. . . . . 401  
*Pavel Klavík and Maria Saumell*

Faster Algorithms for Computing the R\* Consensus Tree . . . . . 414  
*Jesper Jansson, Wing-Kin Sung, Hoa Vu, and Siu-Ming Yiu*

**Fixed-Parameter Tractable Algorithms II**

Complexity and Kernels for Bipartition into Degree-Bounded Induced Graphs. . . . .	429
<i>Mingyu Xiao and Hiroshi Nagamochi</i>	
Faster Existential FO Model Checking on Posets . . . . .	441
<i>Jakub Gajarský, Petr Hliněný, Jan Obdržálek, and Sebastian Ordyniak</i>	
Vertex Cover Reconfiguration and Beyond . . . . .	452
<i>Amer E. Mouawad, Naomi Nishimura, and Venkatesh Raman</i>	

**Graph Algorithms: Approximation I**

Approximating the Maximum Internal Spanning Tree Problem via a Maximum Path-Cycle Cover . . . . .	467
<i>Xingfu Li and Daming Zhu</i>	
Approximation Algorithms Inspired by Kernelization Methods. . . . .	479
<i>Faisal N. Abu-Khzam, Cristina Bazgan, Morgan Chopin, and Henning Fernau</i>	
An 5/4-Approximation Algorithm for Sorting Permutations by Short Block Moves . . . . .	491
<i>Haitao Jiang, Haodi Feng, and Daming Zhu</i>	

**Online and Approximation Algorithms**

Lower Bounds for On-line Graph Colorings. . . . .	507
<i>Grzegorz Gutowski, Jakub Kozik, Piotr Micek, and Xuding Zhu</i>	
An On-line Competitive Algorithm for Coloring $P_8$ -free Bipartite Graphs . . .	516
<i>Piotr Micek and Veit Wiechert</i>	
Bounds on Double-Sided Myopic Algorithms for Unconstrained Non-monotone Submodular Maximization . . . . .	528
<i>Norman Huang and Allan Borodin</i>	

**Data Structures and Algorithms II**

Tradeoff Between Label Space and Auxiliary Space for Representation of Equivalence Classes . . . . .	543
<i>Hicham El-Zein, J. Ian Munro, and Venkatesh Raman</i>	
Depth-First Search Using $O(n)$ Bits. . . . .	553
<i>Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara</i>	
Dynamic Path Counting and Reporting in Linear Space . . . . .	565
<i>Meng He, J. Ian Munro, and Gelin Zhou</i>	

**Matching and Assignment II**

Linear-Time Algorithms for Proportional Apportionment. . . . . 581  
*Zhanpeng Cheng and David Eppstein*

Rank-Maximal Matchings – Structure and Algorithms . . . . . 593  
*Pratik Ghosal, Meghana Nasre, and Prajakta Nimbhorkar*

The Generalized Popular Condensation Problem. . . . . 606  
*Yen-Wei Wu, Wei-Yin Lin, Hung-Lung Wang, and Kun-Mao Chao*

**Graph Algorithms: Approximation II**

Dirichlet Eigenvalues, Local Random Walks, and Analyzing Clusters  
in Graphs. . . . . 621  
*Pavel Kolev and He Sun*

Planar Embeddings with Small and Uniform Faces. . . . . 633  
*Giordano Da Lozzo, Vít Jelínek, Jan Kratochvíl, and Ignaz Rutter*

Scheduling Unit Jobs with a Common Deadline to Minimize the Sum  
of Weighted Completion Times and Rejection Penalties . . . . . 646  
*Nevzat Onur Domaniç and C. Gregory Plaxton*

**Combinatorial Optimization II**

Solving Multi-choice Secretary Problem in Parallel: An Optimal  
Observation-Selection Protocol. . . . . 661  
*Xiaoming Sun, Jia Zhang, and Jialin Zhang*

A Geometric Approach to Graph Isomorphism. . . . . 674  
*Pawan Aurora and Shashank K. Mehta*

Concentrated Hitting Times of Randomized Search Heuristics with  
Variable Drift. . . . . 686  
*Per Kristian Lehre and Carsten Witt*

**Computational Geometry III**

Euclidean TSP with Few Inner Points in Linear Space . . . . . 701  
*Paweł Gawrychowski and Damian Rusak*

Bottleneck Partial-Matching Voronoi Diagrams and Applications. . . . . 714  
*Matthias Henze and Rafel Jaume*

Ham-Sandwich Cuts for Abstract Order Types. . . . . 726  
*Stefan Felsner and Alexander Pilz*

**Network and Scheduling Algorithms**

Graph Orientation and Flows over Time . . . . . 741  
*Ashwin Arulsevan, Martin Groß, and Martin Skutella*

A Simple Efficient Interior Point Method for Min-Cost Flow. . . . . 753  
*Ruben Becker and Andreas Karrenbauer*

Decremental All-Pairs ALL Shortest Paths and Betweenness Centrality . . . . 766  
*Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran*

**Author Index** . . . . . 779

# **Computational Geometry I**

# Line-Constrained $k$ -Median, $k$ -Means, and $k$ -Center Problems in the Plane

Haitao Wang and Jingru Zhang<sup>(✉)</sup>

Department of Computer Science, Utah State University, Logan, UT 84322, USA  
haitao.wang@usu.edu, jingruzhang@aggiemail.usu.edu

**Abstract.** The (weighted)  $k$ -median,  $k$ -means, and  $k$ -center problems in the plane are known to be NP-hard. In this paper, we study these problems with an additional constraint that requires the sought  $k$  facilities to be on a given line. We present efficient algorithms for various distance metrics such as  $L_1$ ,  $L_2$ ,  $L_\infty$ . Assume all  $n$  weighted points are given sorted by their projections on the given line. For  $k$ -median, our algorithms for  $L_1$  and  $L_\infty$  metrics run in  $O(\min\{nk, n\sqrt{k \log n \log n}, n2^{O(\sqrt{\log k \log \log n}) \log n}\})$  time and  $O(\min\{nk \log n, n\sqrt{k \log n \log^2 n}, n2^{O(\sqrt{\log k \log \log n}) \log^2 n}\})$  time, respectively. For  $k$ -means, which is defined only on the  $L_2$  metric, we give an  $O(\min\{nk, n\sqrt{k \log n}, n2^{O(\sqrt{\log k \log \log n})}\})$  time algorithm. For  $k$ -center, our algorithms run in  $O(n \log n)$  time for all three metrics, and in  $O(n)$  time for the unweighted version under  $L_1$  and  $L_\infty$  metrics.

## 1 Introduction

It has been known that the (weighted)  $k$ -median,  $k$ -means, and  $k$ -center in the plane are NP-hard [15, 24, 27]. In this paper, we study these problems with an additional constraint that the sought  $k$  facilities must be on a given line.

For any point  $p$ , denote by  $x(p)$  and  $y(p)$  its  $x$ - and  $y$ -coordinates, respectively. For any two points  $p$  and  $q$ , denote by  $d(p, q)$  the distance between  $p$  and  $q$ . Depending on the distance metrics,  $d(p, q)$  may refer to the  $L_1$  distance, i.e.,  $|x(p) - x(q)| + |y(p) - y(q)|$ , or the  $L_2$  distance, i.e.,  $\sqrt{(x(p) - x(q))^2 + (y(p) - y(q))^2}$ , or the  $L_\infty$  distance, i.e.,  $\max\{|x(p) - x(q)|, |y(p) - y(q)|\}$ . For convenience, we define the  $L_2^2$  distance metric as  $(x(p) - x(q))^2 + (y(p) - y(q))^2$ .

Let  $P$  be a set of  $n$  points in the plane, and each point  $p \in P$  has a weight  $w(p) > 0$ . The goal of the  $k$ -median (resp.,  $k$ -center) problem is to find a set  $Q$  of  $k$  points (called facilities) in the plane such that  $\sum_{p \in P} [w(p) \cdot \min_{q \in Q} d(p, q)]$  (resp.,  $\max_{p \in P} [w(p) \cdot \min_{q \in Q} d(p, q)]$ ) is minimized. The  $k$ -means problem is actually the  $k$ -median problem under the  $L_2^2$  metric.

If all points of  $Q$  are required to be on a given line, denoted by  $\chi$ , then we refer to the corresponding problems as *line-constrained* or simply *constrained*  $k$ -median,  $k$ -means, and  $k$ -center problems. In the following paper, we assume  $\chi$  is the  $x$ -axis and the points of  $P$  have been sorted by their  $x$ -coordinates.

---

This research was supported in part by NSF under Grant CCF-1317143.

**Table 1.** Summary of our results, where  $\tau = \min\{n\sqrt{k \log n}, n2^{O(\sqrt{\log k \log \log n})}\}$ . Furthermore, the unweighted  $L_1$  constrained  $k$ -median is solved in  $O(\tau)$  time. The unweighted  $L_1$  and  $L_\infty$  constrained  $k$ -center are solved in  $O(n)$  time.

	constrained $k$ -median	constrained $k$ -center
$L_1$	$O(\min\{nk, \tau \log n\})$	$O(n \log n)$
$L_\infty$	$O(\min\{nk \log n, \tau \log^2 n\})$	$O(n \log n)$
$L_2^2$	$O(\min\{nk, \tau\})$ (i.e., the constrained $k$ -means)	not applicable

Throughout the paper, let  $\tau = \min\{n\sqrt{k \log n}, n2^{O(\sqrt{\log k \log \log n})}\}$ . See Table 1 for our results. For the constrained  $k$ -median, our algorithms for the  $L_1$  and  $L_\infty$  metrics run in  $O(\min\{nk, \tau \log n\})$  and  $O(\min\{nk \log n, \tau \log^2 n\})$  time, respectively. The  $L_1$  unweighted version where all points of  $P$  have the same weight can be solved in  $O(\tau)$  time. These time bounds almost match those of the best algorithms for the one-dimensional  $k$ -median problems. Note that the  $L_2$  version of the constrained  $k$ -median has been shown unsolvable due to the computation challenge even for  $k = 1$  [5]. For the constrained  $k$ -means, we give an  $O(\min\{nk, \tau\})$  time algorithm. For the constrained  $k$ -center, our algorithms run in  $O(n \log n)$  time for all three metrics, and in  $O(n)$  time for the unweighted version under  $L_1$  and  $L_\infty$  metrics. These  $k$ -center results are optimal.

Our results show that although these problems in 2D are hard, their “1.5D” versions are “easy”. A practical example in which the facilities are restricted to lie along a line is that we want to build some supply centers along a railway or highway (although a railway or highway may not be a straight line, it may be considered straight in each local area). Other relevant examples may include building partial delivery stations along an oil or gas transportation pipeline.

## 1.1 Previous Work

The  $L_1$  and  $L_2$   $k$ -median and  $k$ -center problems in the plane are NP-hard [27], and so as the  $L_\infty$   $k$ -center problem [15]. In the one-dimensional space, however, both problems are solvable in polynomial time: For  $k$ -median, the best-known algorithms run in  $O(nk)$  time [4, 18] or in  $O(\tau \log n)$  time [11]; for  $k$ -center, the best-known algorithms run in  $O(n \log n)$  time [10, 12, 26].

The  $k$ -means problem in the plane is also NP-hard [24]. Heuristic and approximation algorithms have been proposed, e.g., see [13, 20, 23, 29].

The unweighted versions of the constrained  $k$ -center were studied before. The  $L_2$  case was first proposed and solved in  $O(n \log^2 n)$  time by Brass *et al.* [6] and later was improved to  $O(n \log n)$  time by Karmakar *et al.* [21]. Algorithms of  $O(n \log n)$  time were also given in [6] for  $L_1$  and  $L_\infty$  metrics; note that unlike our results, even the points are given sorted, the above algorithms [6] still run in  $O(n \log n)$  time. In addition, Brass *et al.* [6] also gave interesting and efficient algorithms for other two variations of the unweighted  $k$ -center problems, i.e., the line  $\chi$  is not fixed but its slope is fixed, or  $\chi$  is arbitrary. To the best of our knowledge, we are not aware of any previous work on the weighted versions of the constrained  $k$ -median and  $k$ -center problems studied in this paper.



Efficient algorithms have been given for other special cases. When  $k = 1$ , Megiddo [25] solved the unweighted  $L_2$  1-center problem in  $O(n)$  time. Hurtado [19] gave an  $O(n + m)$  time algorithm for the unweighted  $L_2$  1-center problem with the center restricted in a given convex polygon of  $m$  vertices. For  $k = 2$ , Chan [7] proposed an  $O(n \log^2 n \log^2 \log n)$  time for the unweighted  $L_2$  2-center problem and another randomized algorithm; if the points are in convex positions, the same problem can be solved in  $O(n \log^2 n)$  time [22]. The  $L_2$  1-median problem is also known as the Weber problem and no exact algorithm is known for it (and even for the constrained version) [5].

Alt *et al.* [3] studied a somewhat similar problem to our unweighted constrained problems, where the goal is to find a set of disks whose union covers all points and whose centers must be on a given line such that the *sum* of the radii of all disks is minimized, and they gave an  $O(n^2 \log n)$  time algorithm [3]. Note that this problem is different from our  $k$ -median,  $k$ -means, or  $k$ -center problems.

## 1.2 Our Approaches

Suppose  $p_1, p_2, \dots, p_n$  are the points of  $P$  ordered by increasing  $x$ -coordinate. We discover an easy but crucial observation: for every problem studied in this paper, there always exists an optimal solution in which the points of  $P$  “served” by the same facility are consecutive in their index order.

For convenience of discussion, in the following paper we will refer to the  $k$ -means problem as the  $k$ -median problem under the  $L_2^2$  metric.

Based on the above observation, for the constrained  $k$ -median, we propose an algorithmic scheme that works for all metrics (i.e.,  $L_1$ ,  $L_2$ ,  $L_2^2$ , and  $L_\infty$ ), by modeling the problem as finding a minimum weight  $k$ -link path in a DAG  $G$ . Furthermore, we prove that the weights of the edges of  $G$  satisfy the concave Monge property and thus efficient techniques [2, 28] can be used. One challenging problem for the scheme is that we need to design a data structure to compute any graph edge weight (i.e., given any  $i$  and  $j$  with  $i \leq j$ , compute the optimal objective value for the constrained 1-median problem on the points  $p_i, p_{i+1}, \dots, p_j$ ).

For the  $L_2^2$  metric (i.e., the  $k$ -means), we build such a data structure in  $O(n)$  time that can answer each query in  $O(1)$  time. For the  $L_\infty$  metric, we build such a data structure in  $O(n \log n)$  time that can answer each query in  $O(\log^2 n)$  time. Combining this data structure with the above algorithmic scheme, we can solve the  $L_2^2$  and  $L_\infty$  cases. In addition, based on interesting observations, we give another algorithm for the  $L_\infty$  case that is faster than the above scheme for a certain range of values of  $k$ . For the  $L_1$  metric, instead of using the above algorithmic scheme, we reduce the problem to the one-dimensional  $k$ -median problem and then the algorithms in [4, 11, 18] can be applied.

For the constrained  $k$ -center, to solve the  $L_2$  case, we generalize the  $O(n \log n)$  time algorithm in [21] for the unweighted version. In fact, similar approaches can also solve the  $L_1$  and  $L_\infty$  cases. However, since the algorithm uses Cole’s parametric search [12], which is complicated and involves large constants and thus is only of theoretical interest, we design another  $O(n \log n)$  time algorithms for the  $L_1$  and  $L_\infty$  cases, without using parametric search.

In addition, for the unweighted  $L_1$  and  $L_\infty$  cases, due to the above crucial observation, our linear time algorithm hinges on the following efficient data structures. With  $O(n)$  time preprocessing, for any query  $i \leq j$ , we can solve in  $O(1)$  time the constrained  $L_1$  and  $L_\infty$  1-median problems on the points  $p_i, p_{i+1}, \dots, p_j$ .

Note that our algorithms for the  $L_2$  and  $L_2^2$  metrics work for any arbitrary line  $\chi$  (but  $\chi$  must be given as input). However, since the distances under  $L_1$  and  $L_\infty$  metrics are closely related to the orientation of the coordinate system, our algorithms for them only work for horizontal lines  $\chi$ .

We introduce some notations and observations in Section 2. In Sections 3 and 4, we present our algorithms for the constrained  $k$ -median (including the  $k$ -means) and  $k$ -center problems, respectively. Due to the space limit, all lemma and theorem proofs are omitted and can be found in the full paper.

## 2 Preliminaries

For simplicity of discussion, we assume no two points in  $P$  have the same  $x$ -coordinate. Let  $p_1, p_2, \dots, p_n$  be the points of  $P$  ordered by increasing  $x$ -coordinate. Define  $P(i, j) = \{p_i, p_{i+1}, \dots, p_j\}$  for any  $i \leq j$ . For any  $1 \leq i \leq n$ , we also use  $x_i, y_i$ , and  $w_i$  to refer to  $x(p_i), y(p_i)$ , and  $w(p_i)$ , respectively.

For any facility set  $Q$  and any point  $p$ , let  $d(p, Q) = \min_{q \in Q} d(p, q)$ . For any point  $p \in P$ , if  $d(p, Q) = d(p, q)$  for some facility point  $q \in Q$ , then we say  $p$  is “served” by  $q$ . We call  $\sum_{p \in P} [w(p) \cdot d(p, Q)]$  and  $\max_{p \in P} [w(p) \cdot d(p, Q)]$  the *objective value* of the  $k$ -median and  $k$ -center problems, respectively. The following is an easy but crucial lemma.

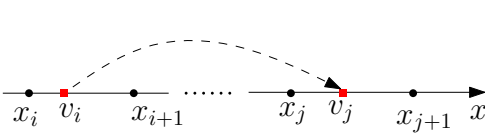
**Lemma 1.** *For each of the constrained  $k$ -median and  $k$ -center problems of any metric (i.e.,  $L_1, L_2, L_2^2$ , or  $L_\infty$ ), there must exist an optimal solution in which the points of  $P$  served by the same facility are consecutive in their index order.*

For any  $i \leq j$ , consider the constrained 1-median problem on  $P(i, j)$ ; denote by  $f(i, j)$  the facility in an optimal solution and define  $\alpha(i, j)$  to be the objective value of the optimal solution, i.e.,  $\alpha(i, j) = \sum_{t=i}^j [w_t \cdot d(p_t, f(i, j))]$ . We call  $f(i, j)$  the *constrained median* of  $P(i, j)$ . In the case that  $f(i, j)$  is not unique, we let  $f(i, j)$  refer to the leftmost such point. By Lemma 1, solving the constrained  $k$ -median problem is equivalent to partitioning the sequence  $p_1, p_2, \dots, p_n$  into  $k$  subsequences such that the sum of the  $\alpha$  values of all these subsequences is minimized. There are also similar observations for the constrained  $k$ -center problem. As will be seen later, these observations are quite useful for our algorithms.

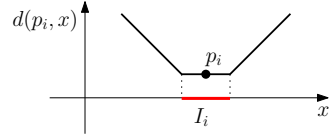
For any point  $p$  on the  $x$ -axis, for convenience, we also use  $p$  to denote its  $x$ -coordinate. For example, if two points  $p$  and  $q$  are on the  $x$ -axis, then  $p < q$  means that  $p$  is strictly to the left of  $q$ . For any value  $x$ , we sometime also use  $x$  to refer to the point on the  $x$ -axis with  $x$ -coordinate  $x$ .

## 3 The Constrained $k$ -Median

This section gives our algorithms for the constrained  $k$ -median under  $L_1, L_2^2$ , and  $L_\infty$  metrics. We first propose an algorithmic scheme in Section 3.1 that



**Figure 1.** Illustrating an edge of  $G$  from  $v_i$  to  $v_j$ , i.e., the two (red) squared points



**Figure 2.** Illustrating the function  $d(p_i, x)$ : the (red) thick segment is  $I_i$

works for any metric. To use the scheme, one has to design a data structure for computing  $\alpha(i, j)$  for any query  $i$  and  $j$  with  $i \leq j$ . We solve the  $L_2^2$  case (i.e., the  $k$ -means) by giving such a data structure in the end of Section 3.1. In Section 3.2, we design such a data structure for  $L_\infty$  metric, and thus solves the  $L_\infty$  case. In addition, we give another algorithm that is faster than the scheme for a certain range of values of  $k$ . Instead of using the scheme, we get a better result for the  $L_1$  case in Section 3.3 by reducing it to the one-dimensional problem.

### 3.1 An Algorithmic Scheme for All Metrics

In this subsection, unless otherwise stated, all notations involving distances, e.g.,  $d(p, q)$ ,  $\alpha(i, j)$ , can use any distance metric (i.e.,  $L_1$ ,  $L_2$ ,  $L_2^2$ , and  $L_\infty$ ).

In light of our observations in Section 2, we will reduce the problem to finding a minimum weight  $k$ -link path in a DAG  $G$ . Further, we will show that the edge weights of  $G$  satisfy the concave Monge property and then efficient algorithms [1, 2, 28] can be used. Below, we first define the graph  $G$ .

For each point  $p_i \in P$ , recall that  $x_i = x(p_i)$  and we also use  $x_i$  to denote the projection of  $p_i$  on the  $x$ -axis. The vertex set of  $G$  consists of  $n + 1$  vertices  $v_0, v_1, \dots, v_n$  and one can consider each  $v_i$  corresponding to a point between  $x_i$  and  $x_{i+1}$  ( $v_0$  is to the left of  $x_1$  and  $v_n$  is to the right of  $x_n$ ); e.g., see Fig.1. For any  $i$  and  $j$  with  $0 \leq i \leq j \leq n$ , we define a directed edge  $e(i, j)$  from  $v_i$  to  $v_j$ , and the weight of the edge, denoted by  $w(i, j)$ , is defined to be  $\alpha(i + 1, j)$  (if we view  $v_i$  and  $v_j$  as two points on the  $x$ -axis as above, then  $\overline{v_i v_j}$  contains the points  $x_{i+1}, x_{i+2}, \dots, x_j$ ). Clearly,  $G$  is a directly acyclic graph (DAG).

A path in  $G$  is a  $k$ -link path if it has  $k$  edges. The *weight* of any path is the sum of the weights of all edges of the path. A *minimum weight  $k$ -link path* from  $v_0$  to  $v_n$  in  $G$  is a  $k$ -link path that has the minimum weight among all  $k$ -link paths from  $v_0$  to  $v_n$ . Note that any  $k$ -link path from  $v_0$  to  $v_n$  in  $G$  corresponds to a partition of the points in  $P$  into  $k$  subsequences. According to our observations in Section 2 and the definition of  $G$ , the following lemma is self-evident.

**Lemma 2.** *A minimum weight  $k$ -link path  $\pi$  from  $v_0$  to  $v_n$  in  $G$  corresponding to an optimal solution  $OPT$  of the constrained  $k$ -median problem on  $P$ . Specifically, the objective value of  $OPT$  is equal to the weight of  $\pi$ , and for each edge  $e(v_i, v_j)$  of  $\pi$ , there is a corresponding facility serving all points of  $P(i + 1, j)$  in  $OPT$ .*

**Lemma 3.** *For any metric, the weights of the edges of  $G$  satisfy the concave Monge property, i.e.,  $w(i, j) + w(i + 1, j + 1) \leq w(i, j + 1) + w(i + 1, j)$  holds for any  $1 \leq i < j \leq n$ .*

By Lemma 3, we can apply the algorithm in [1, 2, 28]. Assuming the weight of each graph edge  $w(i, j)$  can be obtained in  $O(1)$  time, the algorithms in [2] and [28] can compute a minimum weight  $k$ -link path from  $v_0$  to  $v_n$  in  $O(n\sqrt{k \log n})$  time and  $O(n2^{O(\sqrt{\log k \log \log n})})$  time, respectively. Further, as indicated in [2], by using dynamic programming and applying the technique in [1], such a path can also be computed in  $O(nk)$  time. In our problem, to compute each  $w(i, j)$  is essentially to compute  $\alpha(i+1, j)$ . Therefore, we can obtain the following result.

**Theorem 1.** *For any metric, if we can build a data structure in  $O(T)$  time that can compute  $\alpha(i, j)$  in  $O(\sigma)$  time for any query  $i \leq j$ , then we can solve the constrained  $k$ -median problem in  $O(T + \sigma \cdot \min\{nk, \tau\})$  time, where  $\tau = \min\{\sqrt{nk \log n}, n2^{O(\sqrt{\log k \log \log n})}\}$ .*

The following result is an application of our algorithmic scheme in Theorem 1 to the  $L_2^2$  case (i.e., the  $k$ -means).

**Theorem 2.** *For the  $L_2^2$  metric, a data structure can be built in  $O(n)$  time that can answer each  $\alpha(i, j)$  query in  $O(1)$  time. Consequently by Theorem 1 the constrained  $k$ -means problem can be solved in  $O(\min\{nk, \tau\})$  time.*

### 3.2 The Constrained $k$ -Median under the $L_\infty$ -Metric

In this section, all notations related to distances use the  $L_\infty$  metric. We present two algorithms. For the first algorithm, our main goal is to prove Lemma 4. Thus, by Theorem 1, we can solve the  $L_\infty$  case in  $O(\min\{nk, \tau\} \cdot \log^2 n)$  time.

**Lemma 4.** *For the  $L_\infty$  metric, a data structure can be constructed in  $O(n \log n)$  time that can answer each  $\alpha(i, j)$  query in  $O(\log^2 n)$  time.*

For any point  $p_i$ , let  $I_i$  denote the interval on the  $x$ -axis centered at  $x_i$  with length  $|y_i|$  (i.e., the absolute value of the  $y$ -coordinate of  $p_i$ ). Note that the points of  $I_i$  have the same ( $L_\infty$ ) distance to  $p_i$ . Consider  $d(p_i, x)$  as a function of a point  $x$  on the  $x$ -axis. As  $x$  changes from  $-\infty$  to  $+\infty$ ,  $d(p_i, x)$  first decreases and then does not change when  $x \in I_i$  and finally increases (e.g., see Fig. 2). Consider any two indices  $i \leq j$ . Let  $E(i, j)$  be the set of the endpoints of all intervals  $I_t$  for  $i \leq t \leq j$ . For any point  $x$  on the  $x$ -axis, define  $\phi(i, j, x) = \sum_{t=i}^j w_t d(p_t, x)$ . By the definition of  $f(i, j)$ ,  $\phi(i, j, x)$  is minimized at  $x = f(i, j)$  and  $\alpha(i, j) = \phi(i, j, f(i, j))$ . Lemma 5 is crucial for computing  $\alpha(i, j)$ .

**Lemma 5.** *The function  $\phi(i, j, x)$  is a continuous piecewise linear function whose slopes change only at the points of  $E(i, j)$ . Further, there exist two points in  $E(i, j)$ , denoted by  $x'$  and  $x''$  with  $x' \leq x''$  ( $x' = x''$  is possible), such that as  $x$  increases from  $-\infty$  to  $+\infty$ ,  $\phi(i, j, x)$  will strictly decrease when  $x \leq x'$ , and will be constant when  $x \in [x', x'']$ , and will strictly increase when  $x \geq x''$ .*

By Lemma 5, to compute  $\alpha(i, j)$ , which is the minimum value of  $\phi(x, i, j)$ , we can do binary search on the sorted list of  $E(i, j)$ , provided that we can compute

$\phi(i, j, x)$  efficiently for any  $x$ . Since  $E(i, j) \subseteq E(1, n)$ , we can also do binary search on the sorted list of  $E(1, n)$  to compute  $\alpha(i, j)$ . Hence, as preprocessing, we compute the sorted list of  $E(1, n)$  in  $O(n \log n)$  time since  $|E(1, n)| = 2n$ .

According to the above discussion, for any query  $(i, j)$  with  $i \leq j$ , if we can compute  $\phi(i, j, x)$  in  $O(\sigma')$  time for any  $x$ , then we can compute  $\alpha(i, j)$  in  $O(\sigma' \log n)$  time. The following Lemma 6 gives a data structure for answering  $\phi(i, j, x)$  queries, which immediately leads to Lemma 4.

**Lemma 6.** *We can construct a data structure in  $O(n \log n)$  time that can compute  $\phi(i, j, x)$  in  $O(\log n)$  time for any  $i \leq j$  and  $x$ .*

*Proof.* Let  $T$  be a complete binary tree whose leaves correspond to the points of  $P$  from left to right. For each  $1 \leq i \leq n$ , the  $i$ -th leaf is associated with the function  $w_i d(p_i, x)$ , which is actually  $\phi(i, i, x)$ . Consider any internal node  $v$ . Let the leftmost (resp., rightmost) leaf of the subtree rooted at  $v$  be the  $i$ -th (resp.,  $j$ -th) leaf. We associate with  $v$  the function  $\phi(i, j, x)$ , and we also use  $\phi_v(x)$  to denote the function. By Lemma 5, the combinatorial complexity of the function  $\phi_v(x)$  is  $O(j - i + 1)$ . Let  $u$  and  $w$  be  $v$ 's two children. Suppose we have already computed the two functions  $\phi_u(x)$  and  $\phi_w(x)$ ; since essentially  $\phi_v(x) = \phi_u(x) + \phi_w(x)$ , we can easily compute  $\phi_v(x)$  in  $O(j - i + 1)$  time. Therefore, we can compute the tree  $T$  in  $O(n \log n)$  time in a bottom-up fashion.

Consider any query  $i \leq j$  and  $x = x'$  and the goal is to compute  $\phi(i, j, x')$ . By standard approaches, we first find  $O(\log n)$  maximum subtrees such that the leaves of these subtrees are exactly the leaves from the  $i$ -th leaf to the  $j$ -th one. Let  $V$  be the set of the roots of these subtrees. Notice that  $\phi(i, j, x') = \sum_{v \in V} \phi_v(x')$ . For each  $v \in V$ , we can compute the value  $\phi_v(x')$  in  $O(\log n)$  time by doing binary search on the function  $\phi_v(x)$  associated with  $v$ . In this way, since  $|V| = O(\log n)$ , we can compute  $\phi(i, j, x')$  in overall  $O(\log^2 n)$  time. We can avoid doing binary search on every node of  $V$  by constructing a fractional cascading structure [8] on the functions  $\phi_v(x)$  of the nodes of  $T$ . Using fractional cascading, we only need to do one binary search on the root of  $T$ , and then the values  $\phi_v(x')$  for all nodes  $v$  of  $V$  can be computed in constant time each. The fractional cascading structure can be built in additional  $O(n \log n)$  time [8].

As a summary, we can construct a data structure in  $O(n \log n)$  time that can compute  $\phi(i, j, x)$  in  $O(\log n)$  time for any  $i \leq j$  and  $x$ .  $\square$

By Theorem 1 and Lemma 4, we can solve the constrained  $k$ -median problem under  $L_\infty$  metric in  $O(nk \log^2 n)$  or  $O(\tau \log^2 n)$  time.

Our second algorithm is based on the following Lemma 7, which can be easily proved based on Lemma 5.

**Lemma 7.** *For the  $L_\infty$  constrained  $k$ -median problem on  $P$ , there must exist an optimal solution in which the facility set  $Q$  is a subset of  $E(1, n)$ .*

By Lemma 7, we have a set of ‘‘candidate’’ facilities, and further, by Lemma 1, we only need to check these candidates from left to right. Based on these observations, we develop a dynamic programming algorithm and the result is given in Lemma 8.

**Lemma 8.** *The  $L_\infty$  constrained  $k$ -median is solvable in  $O(nk \log n)$  time.*

Combining the two algorithms, we obtain the following theorem.

**Theorem 3.** *The constrained  $k$ -median problem under the  $L_\infty$  metric can be solved in  $O(\min\{nk \log n, \tau \log^2 n\})$  time.*

### 3.3 The Constrained $k$ -Median Problem under $L_1$ -Metric

To solve the  $L_1$  case, instead of using Theorem 1, we get a better result by reducing it to the one-dimensional problem and then applying the algorithms in [4, 11, 18]. In this section, all notations related to distances use the  $L_1$  metric.

Recall that our goal is to minimize  $\sum_{i=1}^n [w_i \cdot d(p_i, Q)]$ . Consider any point  $p_i \in P$ . For any point  $q$  on the  $x$ -axis, since  $d(p_i, q)$  is the  $L_1$  distance, we have  $d(p_i, q) = d(x_i, q) + |y_i|$ . Since all points of  $Q$  are on the  $x$ -axis, it holds that  $d(p_i, Q) = \min_{q \in Q} d(p_i, q) = |y_i| + \min_{q \in Q} d(x_i, q)$ . Therefore, we obtain  $\sum_{i=1}^n [w_i \cdot d(p_i, Q)] = \sum_{i=1}^n w_i |y_i| + \sum_{i=1}^n [w_i \cdot d(x_i, Q)]$ .

Note that once  $P$  is given,  $\sum_{i=1}^n w_i |y_i|$  is constant, and thus, to minimize  $\sum_{i=1}^n [w_i \cdot d(p_i, Q)]$  is equivalent to minimizing  $\sum_{i=1}^n [w_i \cdot d(x_i, Q)]$ , which is essentially the following *one-dimensional  $k$ -median problem*: Given a set of  $n$  points  $P' = \{x_1, x_2, \dots, x_n\}$  on the  $x$ -axis with each  $x_i$  having a weight  $w(x_i) = w_i \geq 0$ , find a set  $Q$  of  $k$  points on the  $x$ -axis to minimize  $\sum_{i=1}^n [w_i \cdot d(x_i, Q)]$ .

The above 1D  $k$ -median problem is a *continuous version* because each point of our facility set  $Q$  can be any point on the  $x$ -axis. There is also a *discrete version*, where  $Q$  is required to be a subset of  $P'$ . The algorithms given in [4, 11, 18] are for the discrete version and therefore we cannot apply their algorithms directly. Fortunately, due to some observations, we prove below that for our continuous version there always exists an optimal solution in which the set  $Q$  is a subset of  $P'$ , and consequently we can apply the discrete version algorithms.

Consider any indices  $i \leq j$ . Let  $P'(i, j) = \{x_i, x_{i+1}, \dots, x_j\}$ . As in the  $L_\infty$  case, for any point  $x$  on the  $x$ -axis, define  $\phi(i, j, x) = \sum_{t=i}^j w_t d(x_t, x)$ . The following lemma is similar in spirit to Lemma 5.

**Lemma 9.** *The function  $\phi(i, j, x)$  is a continuous piecewise linear function whose slopes change only at the points of  $P'(i, j)$ . Further, there exist two points in  $P'(i, j)$ , denoted by  $x'$  and  $x''$  with  $x' \leq x''$  ( $x' = x''$  is possible), such that as  $x$  increases from  $-\infty$  to  $+\infty$ ,  $\phi(i, j, x)$  will strictly decrease when  $x \leq x'$ , and will be constant when  $x \in [x', x'']$ , and will strictly increase when  $x \geq x''$ .*

By Lemma 9, we can obtain the following lemma.

**Lemma 10.** *For the 1D  $k$ -median problem on  $P'$ , there must exist an optimal solution in which the facility set  $Q$  is a subset of  $P'$ .*

In light of Lemma 10, we can apply the algorithms in [4, 11, 18] to solve the  $k$ -median problem on  $P'$ . The algorithms in [4, 18] run in  $O(nk)$  time and the algorithm in [11] runs in  $O(\tau \log n)$  time and  $O(\tau)$  time for the unweighted case.

**Theorem 4.** *The  $L_1$  constrained  $k$ -median can be solved in  $O(\min\{nk, \tau \log n\})$  time and the unweighted case can be solved in  $O(\min\{nk, \tau\})$  time.*

## 4 The Constrained $k$ -Center

This section presents our  $k$ -center algorithms. We first give a linear time algorithm to solve the decision version of the problem for all metrics. Then, we present an  $O(n \log n)$  time algorithm for the  $L_2$  metric. In fact, similar algorithms also work for the other two metrics. However, since the algorithm uses Cole’s parametric search [12], which is complicated, we give another  $O(n \log n)$  time algorithm for  $L_1$  and  $L_\infty$  metrics, without using parametric search. Finally, we give an  $O(n)$  time algorithm for the unweighted case under  $L_1$  and  $L_\infty$  metrics.

In the following, unless otherwise stated, all notations related to distances are applicable to all three metrics, i.e.,  $L_1$ ,  $L_2$ , and  $L_\infty$ .

The *decision version* of the problem is as follows: given any value  $\epsilon$ , determine whether there are a set  $Q$  of  $k$  facilities such that  $\max_{p \in P}[w(p) \cdot d(p, Q)] \leq \epsilon$ , and if yes, we call  $\epsilon$  a *feasible value*. We let  $\epsilon^*$  denote the optimal objective value, i.e.,  $\epsilon^* = \max_{p \in P}[w(p) \cdot d(p, Q)]$  for the facility set  $Q$  in any optimal solution. Hence, for any  $\epsilon$ , it is a feasible value if and only  $\epsilon \geq \epsilon^*$ .

For any point  $p_i \in P$ , denote by  $I(p_i, \epsilon)$  the set of points  $q$  on the  $x$ -axis such that  $w_i d(p_i, q) \leq \epsilon$ . Note that  $I(p_i, \epsilon)$  is the intersection of the “disk” centered at  $p_i$  with radius  $\epsilon/w_i$  (the “disk” is a diamond, a real circular disk, and a square under  $L_1$ ,  $L_2$ , and  $L_\infty$  metrics, respectively). Hence,  $I(p_i, \epsilon)$  is an interval and we refer to  $I(p_i, \epsilon)$  as the *facility location interval* of  $p_i$ . Note that for any subset  $P(i, j)$ , if the intersection of all facility location intervals of  $P(i, j)$  is not empty, then any point in the above intersection can be used as a facility to serve all points of  $P(i, j)$  within weighted distance  $\epsilon$ .

We say a point *covers* an interval on the  $x$ -axis if the interval contains the point. Let  $I(P, \epsilon)$  be the set of all facility location intervals of  $P$ . According to the above discussion, to determine whether  $\epsilon$  is a feasible value, it is sufficient to compute a minimum number of points that can cover all intervals of  $I(P, \epsilon)$ , which can be done in  $O(n)$  time after the endpoints of all intervals of  $I(P, \epsilon)$  are sorted [17]. The overall time for solving the decision problem is  $O(n \log n)$  due to the sorting. Below, we give an  $O(n)$  time algorithm, without sorting.

Similar to Lemma 1, if  $\epsilon$  is a feasible value, then there exists a feasible solution in which each facility serves a set of consecutive points of  $P$ . Using this observation, our algorithm works as follows. We consider the intervals of  $I(P, \epsilon)$  from  $I(p_1, \epsilon)$  in the index order of  $p_i$ . We find the largest index  $j$  such that  $\bigcap_{i=1}^j I(p_i, \epsilon)$  is not empty, and then we place a facility at any point in the above intersection to serve all points in  $P(1, j)$ . Next, from  $I(p_{j+1}, \epsilon)$ , we find the next maximal subset of intervals whose intersection is not empty to place a facility. We continue this procedure until the last interval  $I(p_n, \epsilon)$  has been considered. Clearly, the running time of the algorithm is  $O(n)$ . Let  $k'$  be the number of facilities that are placed in the above procedure. The value  $\epsilon$  is a feasible value if and only if  $k' \leq k$ . Hence, we have the following result.

**Lemma 11.** *Given any value  $\epsilon$ , we can determine whether  $\epsilon$  is a feasible value in  $O(n)$  time for any metric.*

#### 4.1 The $L_2$ Metric

For any  $\epsilon$  and each  $1 \leq i \leq n$ , let  $l_i(\epsilon)$  and  $r_i(\epsilon)$  denote the left and right endpoints of  $I(p_i, \epsilon)$ , respectively. Recall that  $\epsilon^*$  is the optimal objective value. Let  $S = \{l_i(\epsilon^*), r_i(\epsilon^*) \mid 1 \leq i \leq n\}$ . If we know the sorted lists of the values of  $S$ , then we can use our decision algorithm to compute an optimal facility set in  $O(n)$  time. Although we do not know  $\epsilon^*$ , we can still sort  $S$  by parametric search [12]. In the parametric search, we will need to compare two values of  $S$ . Although we do not know  $\epsilon^*$ , we can resolve the comparison using our decision algorithm in Lemma 11. We omit the details.

**Theorem 5.** *The constrained  $k$ -center problem under the  $L_2$  metric can be solved in  $O(n \log n)$  time, by using Cole's parametric search.*

#### 4.2 The $L_1$ and $L_\infty$ Metrics

We present  $O(n \log n)$  time algorithms for the  $L_1$  and  $L_\infty$  metrics, without using parametric search. We consider the  $L_1$  case first.

We define  $l_i(\epsilon)$  and  $r_i(\epsilon)$  in the same way as before. We consider  $l_i(\epsilon)$  and  $r_i(\epsilon)$  as functions of  $\epsilon$ . It can be verified that  $r_i(\epsilon) = x_i + \epsilon/w_i - |y_i|$  and  $l_i(\epsilon) = x_i - \epsilon/w_i + |y_i|$ , both defined on  $\epsilon \geq w_i \cdot |y_i|$ . Hence, each of  $l_i(\epsilon)$  and  $r_i(\epsilon)$  defines a half-line. Let  $A$  be the set of the half-lines defined by  $l_i(\epsilon)$  and  $r_i(\epsilon)$  for all  $i = 1, 2, \dots, n$ . As analyzed in [6] for the unweighted case, the optimal objective value  $\epsilon^*$  must be the  $y$ -coordinate of an intersection of two half-lines of  $A$ . In fact,  $\epsilon^*$  is the smallest feasible value among the  $y$ -coordinates of all intersections of the half-lines of  $A$ . Let  $\mathcal{A}$  be the arrangement of the lines containing the half-lines of  $A$ . The intersection of two lines of  $\mathcal{A}$  is called a *vertex*. Hence,  $\epsilon^*$  is the smallest feasible value among the  $y$ -coordinates of the vertices of  $\mathcal{A}$ . Therefore, to solve the constrained  $k$ -center problem on  $P$ , it is sufficient to find the lowest vertex (denoted by  $v^*$ ) of  $\mathcal{A}$  whose  $y$ -coordinate is a feasible value (which is  $\epsilon^*$ ) and then apply our decision algorithm in Lemma 11 on  $\epsilon^*$  to find an optimal facility set in additional  $O(n)$  time. Such a vertex  $v^*$  can be found by using a line arrangement searching technique given in [9] and our decision algorithm. The details are omitted.

**Lemma 12.** *Such a vertex  $v^*$  can be found in  $O(n \log n)$  time.*

Hence, we can solve the  $L_1$  constrained  $k$ -center problem in  $O(n \log n)$  time.

For the  $L_\infty$  case, the algorithm is similar. Under  $L_\infty$  metric, it can be verified that  $r_i(\epsilon) = x_i + \epsilon/w_i$  and  $l_i(\epsilon) = x_i - \epsilon/w_i$ , both defined on  $\epsilon \geq w_i \cdot |y_i|$ . Hence, each of  $r_i(\epsilon)$  and  $l_i(\epsilon)$  still defines a half-line, as in the  $L_1$  case. Therefore, we can use the similar algorithm as in the  $L_1$  case.

**Theorem 6.** *The  $L_1$  and  $L_\infty$  constrained  $k$ -center problems can be solved in  $O(n \log n)$  time, without using parametric search.*



### 4.3 The Unweighted Case under $L_1$ and $L_\infty$ Metrics

We give an  $O(n)$  time algorithm for the unweighted case under  $L_1$  and  $L_\infty$  metrics. For any  $i \leq j$ , consider the constrained 1-center problem on the points in  $P(i, j)$ ; denote by  $g(i, j)$  the facility in an optimal solution and define  $\beta(i, j)$  to be the objective value of the optimal solution, i.e.,  $\beta(i, j) = \max_{i \leq t \leq j} w_t d(p_t, g(i, j))$ . We call  $g(i, j)$  the *constrained center* of  $P(i, j)$ .

By Lemma 1, solving the constrained  $k$ -median problem is equivalent to partitioning the sequence  $p_1, p_2, \dots, p_n$  into  $k$  subsequences such that the maximum of the  $\beta$  values of all these subsequences is minimized. Formally, we want to find  $k - 1$  indices  $i_0 < i_1 < i_2 < \dots < i_{k-1} < i_k$ , with  $i_0 = 0$  and  $i_k = n$ , such that  $\max_{j=1}^k \beta(i_{j-1} + 1, i_j)$  is minimized. This is exactly the MIN-MAX PARTITION problem proposed in [14]. Based on Frederickson's algorithm [16], the following result is a re-statement of Theorem 2 in [14] with respect to our problem.

**Lemma 13.** [14] *If  $\beta(i, j) \leq \beta(i', j')$  holds for any  $1 \leq i' \leq i \leq j \leq j' \leq n$ , then we have the following result. For any metric, suppose after  $O(T)$  time preprocessing, we can compute  $\beta(i, j)$  in  $O(\sigma)$  time for any query  $i \leq j$ ; then the constrained  $k$ -center problem can be solved in  $O(T + n\sigma)$  time.*

Clearly, the condition on  $\beta$  values in Lemma 13 holds for our problem. In Lemma 14, we give data structures for  $\beta(i, j)$  queries under  $L_1$  and  $L_\infty$  metrics.

**Lemma 14.** *For  $L_1$  and  $L_\infty$  metrics, with  $O(n)$  time preprocessing, we can compute  $\beta(i, j)$  in constant time for any query  $i \leq j$ .*

Our linear time algorithm follows immediately from Lemmas 13 and 14.

## References

1. Aggarwal, A., Klawe, M., Moran, S., Shor, P., Wilbur, R.: Geometric applications of a matrix-searching algorithm. *Algorithmica* **2**, 195–208 (1987)
2. Aggarwal, A., Schieber, B., Tokuyama, T.: Finding a minimum weight  $k$ -link path in graphs with concave monge property and applications. *Discrete and Computational Geometry* **12**, 263–280 (1994)
3. Alt, H., Arkin, E., Brönnimann, H., Erickson, J., Fekete, S., Knauer, C., Lenchner, J., Mitchell, J., Whittlesey, K.: Minimum-cost coverage of point sets by disks. In: Proc. of the 22nd Annual Symposium on Computational Geometry, SoCG, pp. 449–458 (2006)
4. Auletta, V., Parente, D., Persiano, G.: Placing resources on a growing line. *Journal of Algorithms* **26**(1), 87–100 (1998)
5. Bajaj, C.: The algebraic degree of geometric optimization problems. *Discrete and Computational Geometry* **3**, 177–191 (1988)
6. Brass, P., Knauer, C., Na, H.S., Shin, C.S., Vigneron, A.: The aligned  $k$ -center problem. *International Journal of Computational Geometry and Applications* **21**, 157–178 (2011)
7. Chan, T.: More planar two-center algorithms. *Computational Geometry: Theory and Applications* **13**(3), 189–198 (1999)

8. Chazelle, B., Guibas, L.: Fractional cascading: I. A Data structuring technique. *Algorithmica* **1**(1), 133–162 (1986)
9. Chen, D., Wang, H.: A note on searching line arrangements and applications. *Information Processing Letters* **113**, 518–521 (2013)
10. Chen, D., Li, J., Wang, H.: Efficient algorithms for one-dimensional  $k$ -center problems (2013), [arXiv:1301.7512](https://arxiv.org/abs/1301.7512)
11. Chen, D., Wang, H.: New algorithms for facility location problems on the real line. *Algorithmica* **69**, 370–383 (2014)
12. Cole, R.: Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM* **34**(1), 200–208 (1987)
13. Floyd, S.: Least squares quantization in PCM. *IEEE Transactions on Information Theory* **28**, 129–137 (1982)
14. Fournier, H., Vigneron, A.: Fitting a step function to a point set. *Algorithmica* **60**(1), 95–109 (2011)
15. Fowler, R., Paterson, M., Tanimoto, S.: Optimal packing and covering in the plane are NP-complete. *Information Processing Letters* **12**, 133–137 (1981)
16. Frederickson, G.: Optimal algorithms for tree partitioning. In: Proc. of the 2nd Annual Symposium of Discrete Algorithms, SODA, pp. 168–177 (1991)
17. Gupta, U., Lee, D., Leung, J.: Efficient algorithms for interval graphs and circular-arc graphs. *Networks* **12**, 459–467 (1982)
18. Hassin, R., Tamir, A.: Improved complexity bounds for location problems on the real line. *Operations Research Letters* **10**, 395–402 (1991)
19. Hurtado, F., Sacristn, V., Toussaint, G.: Some constrained minimax and maximin location problems. *Studies in Locational Analysis* **5**, 17–35 (2000)
20. Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., Wu, A.: A local search approximation algorithm for  $k$ -means clustering. *Computational Geometry: Theory and Applications* **28**, 89–112 (2004)
21. Karmakar, A., Das, S., Nandy, S., Bhattacharya, B.: Some variations on constrained minimum enclosing circle problem. *Journal of Combinatorial Optimization* **25**(2), 176–190 (2013)
22. Kim, S.K., Shin, C.-S.: Efficient algorithms for two-center problems for a convex polygon. In: Du, D.-Z., Eades, P., Sharma, A.K., Lin, X., Estivill-Castro, V. (eds.) *COCOON 2000*. LNCS, vol. 1858, pp. 299–309. Springer, Heidelberg (2000)
23. Kumar, A., Sabharwal, Y., Sen, S.: A simple linear time  $(1 + \epsilon)$ -approximation algorithm for  $k$ -means clustering in any dimensions. In: Proc. of the 45th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 454–462 (2004)
24. Mahajan, M., Nimbhorkar, P., Varadarajan, K.: The planar  $k$ -means problem is NP-hard. *Theoretical Computer Science* **442**, 13–21 (2012)
25. Megiddo, N.: Linear-time algorithms for linear programming in  $R^3$  and related problems. *SIAM Journal on Computing* **12**(4), 759–776 (1983)
26. Megiddo, N.: Linear programming in linear time when the dimension is fixed. *Journal of the ACM* **31**(1), 114–127 (1984)
27. Megiddo, N., Supowit, K.: On the complexity of some common geometric location problems. *SIAM Journal on Computing* **13**, 182–196 (1984)
28. Schieber, B.: Computing a minimum weight  $k$ -link path in graphs with the concave monge property. *Journal of Algorithms* **29**(2), 204–222 (1998)
29. de la Vega, W.F., Karpinski, M., Kenyon, C., Rabani, Y.: Approximation schemes for clustering problems. In: Proc. of the 25th Annual ACM Symposium on Theory of Computing (STOC), pp. 50–58 (2003)

# Reconstructing Point Set Order Types from Radial Orderings

Oswin Aichholzer<sup>1</sup>, Jean Cardinal<sup>2</sup>, Vincent Kusters<sup>3</sup>(✉),  
Stefan Langerman<sup>2</sup>, and Pavel Valtr<sup>4</sup>

<sup>1</sup> Institute for Software Technology, Graz University of Technology,  
Graz, Austria

`oaich@ist.tugraz.at`

<sup>2</sup> Computer Science Department, Université libre de Bruxelles (ULB),  
Brussels, Belgium

`jcardin@ulb.ac.be, slanger@ulb.ac.be`

<sup>3</sup> Department of Computer Science, ETH Zürich, Zurich, Switzerland

`vincent.kusters@inf.ethz.ch`

<sup>4</sup> Department of Applied Mathematics, Charles University,  
Prague, Czech Republic

`valtr@kam.mff.cuni.cz`

**Abstract.** We consider the problem of reconstructing the combinatorial structure of a set of  $n$  points in the plane given partial information on the relative position of the points. This partial information consists of the radial ordering, for each of the  $n$  points, of the  $n - 1$  other points around it. We show that this information is sufficient to reconstruct the chirotope, or labeled order type, of the point set, provided its convex hull has size at least four. Otherwise, we show that there can be as many as  $n - 1$  distinct chirotopes that are compatible with the partial information, and this bound is tight. Our proofs yield polynomial-time reconstruction algorithms. These results provide additional theoretical insights on previously studied problems related to robot navigation and visibility-based reconstruction.

## 1 Introduction

Many properties of point sets in the plane do not depend on the exact coordinates of the points but only on their relative positions. The *order type*, or *chirotope*, of a point set  $P \subset \mathbb{R}^2$  is the orientation (clockwise or counterclockwise) of every ordered triple of  $P$  [1]. More precisely, a chirotope  $\chi$  associates a sign

---

O. Aichholzer is partially supported by the ESF EUROCORES programme EuroGIGA, CRP ComPoSe, Austrian Science Fund (FWF): I648-N18. J. Cardinal is partially supported by the ESF EUROCORES programme EuroGIGA, CRP ComPoSe, and the Fonds National de la Recherche Scientifique (F.R.S. - FNRS). V. Kusters is partially supported by the ESF EUROCORES programme EuroGIGA, CRP GraDR and the Swiss National Science Foundation, SNF Project 20GG21-134306. Part of the work was done during an ESF EUROCORES-funded visit of V. Kusters to J. Cardinal. S. Langerman is Directeur de Recherches du F.R.S.-FNRS.

$\chi(a, b, c) \in \{0, +1, -1\}$  with each ordered triple  $(a, b, c)$  of points, indicating whether the three points  $a, b, c$  make a left turn (+1), a right turn (-1), or are collinear (0). When  $\chi(a, b, c) \neq 0$  for all triples  $(a, b, c)$ , the order type is said to be *uniform* or to be in *general position*. We consider only uniform order types.

Chirotopes must satisfy a collection of well-studied axioms which define the *abstract order types*. For details on the axioms, we refer the reader to a book by Knuth [2], who refers to chirotopes as CC-systems. These axioms form one of the several axiom systems that define *uniform acyclic rank-3 oriented matroids* [3]. An abstract order type  $\chi$  is *realizable* if there exists a point set in  $\mathbb{R}^2$  with order type  $\chi$ . An abstract order type  $\chi$  is typically identified with its opposite  $-\chi$ , where all signs are reversed, and we follow this convention in this paper. Abstract order types correspond exactly to arrangements of pseudolines, as a consequence of the Folkman-Lawrence topological representation theorem [4]. The smallest non-realizable order type corresponds to the well-known Pappus arrangement of nine pseudolines; all smaller order types are realizable. The convex hull  $h_1, \dots, h_t$  of  $\chi$  is uniquely defined (also for non-realizable order types) by the property<sup>1</sup> that  $\chi(h_i, h_{i+1}, v) = +1$  for all  $v \in V \setminus \{h_i, h_{i+1}\}$  and all  $1 \leq i \leq t$ .

Unlike most other publications on order types, we consider *labeled* order types, not order type isomorphism classes. For instance, whereas there is only one order type isomorphism class for four points in convex position, there are actually three such labeled order types. More precisely, given two order types  $\chi_1$  and  $\chi_2$  on a set  $V$ , we define  $\chi_1 = \chi_2$  if and only if either (i) for all  $u, v, w \in V$ :  $\chi_1(u, v, w) = \chi_2(u, v, w)$  or (ii) for all  $u, v, w \in V$ :  $\chi_1(u, v, w) = -\chi_2(u, v, w)$ .

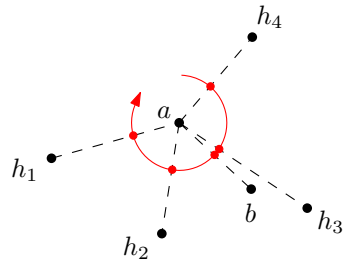
**Radial Orderings and Radial Systems.** We

next introduce the *clockwise radial system*  $R_\chi$  of an abstract order type  $\chi$  (in general position) on a set  $V$ . For an element  $u$  of  $V$ , let  $R_\chi(u)$  be the *clockwise radial ordering* of  $u$ , defined as the unique *cyclic* ordering  $v_1, \dots, v_{n-1}$  of all elements other than  $u$ , sorted clockwise around  $u$ . Figure 1 shows a point set and the clockwise radial orderings of one of its points.

When given only the abstract order type  $\chi$ , we can compute  $R_\chi(u)$  as follows. Let  $v$  be any vertex other than  $u$ . Now sort  $V \setminus \{u\}$  radially around  $u$  by using  $w < w'$  iff  $\chi(u, v, w) > \chi(u, v, w')$ , or  $\chi(u, v, w) = \chi(u, v, w')$  and  $\chi(w, u, w') = +1$  (where  $\chi(u, v, v) := 0$ ).

We write  $U \sim R_\chi$  and say that  $U$  and  $R_\chi$  are *equivalent* if  $U$  can be obtained from  $R_\chi$  by reversing of some of the clockwise radial orderings of  $R_\chi$ . Thus the relation  $\sim$  forgets about the directions of the radial orderings. We call  $U$  an *undirected radial system*, and each  $U(v)$  an *undirected radial ordering*.

While  $\chi$  uniquely determines the equivalence class of  $R_\chi$ , the converse is not necessarily true. We define  $T(U)$  as the set of labeled order types  $\chi$  for which



**Fig. 1.** A point set with  $R_\chi(a) = h_4, h_3, b, h_2, h_1$

<sup>1</sup> Index additions and subtractions are always modulo the length of the sequence.

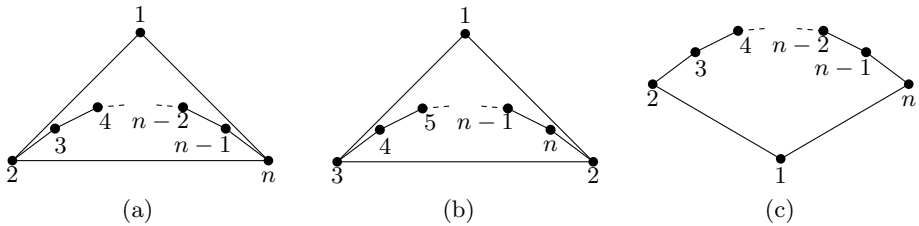
$U \sim R_\chi$ . In this paper we investigate the properties of  $T(U)$ . We show that in many cases  $T(U) = \{\chi\}$  for  $U \sim R_\chi$ : in other words, that  $\chi$  can be reconstructed uniquely from one of its undirected radial systems. However, this is not true in general, as we will discuss below.

**Local Sequences.** Radial orderings are similar in flavor, but different than *local sequences* defined by Goodman and Pollack [5]. The radial ordering around a point  $p$  can be thought of as the order of the intersections of a ray of origin  $p$  with the other points. If instead of a ray, we consider the successive intersections of a rotating *line* through  $p$  with the other points, we get what Goodman and Pollack call the local sequences. The order type (up to projective transformations) can be recovered from the local sequences. Felsner [6] and Felsner and Valtr [7] study simplified encodings of local sequences to prove upper bounds on the number of pseudoline arrangements.

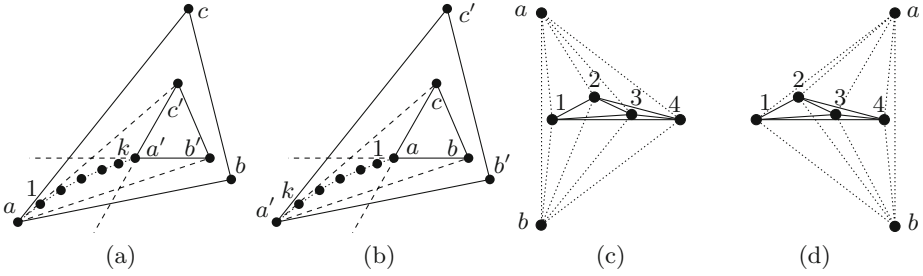
**Examples.** Figure 2 shows three point sets with different (labeled) order types. Figure 2(a) and 2(b) have equivalent radial systems, but Figure 2(c) has a different radial system. Conversely, Figure 2(a) and 2(c) have equivalent local sequences (the sequence for point 1 is reversed), but Figure 2(b) has different local sequences. It follows that local sequences and radial orderings are incomparable in the sense that neither can be computed from the other in general. Figure 2(b) is obtained from Figure 2(a) by cyclically shifting the labels  $2, 3, \dots, n$  once. Each such cyclic shift in this example preserves the undirected radial system  $U$ , and hence  $|T(U)| \geq n - 1$ . We show in what follows that this is the worst case in the sense that  $|T(U)| \leq n - 1$  for all radial systems  $U$ . Figure 3(a-b) shows another example of two point sets with different order types but the same radial system  $U$ . In this case, a discussion later in the paper shows that  $|T(U)| = 2$ .

In the preceding examples, the labeled ordered types were distinct, but isomorphic in the sense they differ only by a relabeling of the points. Figure 3(c-d) shows that this is not always the case: the two point sets have the same radial system and distinct and non-isomorphic order types (see [8]). This construction can be generalized to obtain examples with an arbitrary number of points.

**Related Work.** Concepts similar to radial systems have been studied in a wide variety of contexts. Tovar, Freda and LaValle [9] considered the problem of exploring an unknown environment using a robot that is able to sense the



**Fig. 2.** An example to illustrate the difference between local sequences and radial systems



**Fig. 3.** (a-b) Two point sets with equivalent radial systems. The points  $a, 1, \dots, k, a'$  lie on a convex arc in both sets. (c-d) Two point sets with the same radial system but nonisomorphic order types.

radial orderings of landmarks around it. They use the order type machinery as well, and consider robots with operations like moving towards a landmark to accomplish several recognition tasks. Wismath [10] considered related reconstruction problems involving partial visibility information. He mentions the fact that radial orderings are not always sufficient to reconstruct order types, and solves a related reconstruction problem where, additionally, the  $x$ -coordinate of every point is given. Another similarly flavored problem, the *polygon reconstruction problem from angles*, has been tackled by Disser et al. [11], and Chen and Wang [12]. There they reconstruct a polygon given, for each vertex  $v$ , the sequence of angles formed by the vertices visible from  $v$ . The results developed in this paper will hopefully lay the ground for a complete theoretical treatment of the relation between observed radial orderings and the structure of point sets, and could be useful in such applications.

Some other problems involving radial orderings have been studied in several previous publications. For instance, Devillers et al. [13] considered the problem of maintaining the radial ordering associated with a moving point. Díaz-Báñez, Fabila, and Pérez-Lantero [14] study the number of distinct radial orderings that can be obtained from a point set, and introduce a colored version of the problem. Durocher et al. [15] propose algorithms for realizing radial orderings in point sets. The notion of radial ordering has been used previously by a subset of the current authors in the context of graph drawing. More precisely, it is instrumental in an elementary proof of the  $\exists\text{IR}$ -completeness of the general simultaneous geometric graph embedding problem [16]. Pilz and Welzl [8] consider crossing-preserving mappings between order types. Non-isomorphic order types having the same radial system form an equivalence class in their hierarchy.

**Our Results.** In Section 2, we give a preliminary analysis of radial systems on five points, which will serve as a building block for later sections. In Section 3, we show that  $T(U)$  can be computed from  $U$  in polynomial time. The main procedure involved in the recognition algorithm consists of repeatedly considering five-point configurations, and removing the points that are inside the convex hull of four others. As a byproduct, we can show that if the convex hull has

at least four vertices, then there is at most one compatible order type, that is,  $|T(U)| = 1$ . In Section 4, we prove that  $|T(U)| \leq |V| - 1$  for all undirected rotation systems  $U$  on the set  $V$ . As a consequence of Section 3, this can happen only when the convex hull of the reconstructed order type is a triangle. This bound is tight, as shown by the example of Figure 2(a)-2(b).

For the sake of readability, the proofs involve Euclidean point sets, but we are careful to use only those properties of point sets that hold also for arbitrary order types (realizable or not). An easy way to verify this is to use the representation of abstract order types as *generalized configurations*, discussed in detail in [5]. A generalized configuration in general position is a pair  $(P, L)$  where  $P \subset \mathbb{R}^2$  and  $L$  is a pseudoline arrangement such that every pseudoline in  $L$  contains exactly two points of  $P$ . Note that for realizable order types, such a generalized configuration is obtained simply by taking a point set realization of the order type and its set of supporting lines. Whereas for point sets  $P$ , every triple  $p_1, p_2, p_3 \in P$  defines a cone at  $p_2$ , every triple defines a pseudocone at  $p_2$  (an infinite region bounded by two curves that intersect only at  $p_2$ ) in a generalized configuration, and these have all the properties required for the proofs. Hence, our results extend to abstract order types.

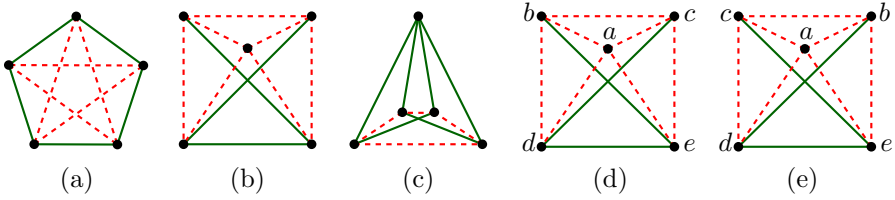
## 2 Bootstrapping

First, we define *signature graphs*, which will prove to be a useful tool in the analysis of undirected radial systems. Given a vertex set  $V$  and some  $U \sim R_\chi$  on  $V$  for some labeled abstract order type  $\chi$ , we construct a labeling of the complete digraph  $D_U$  on  $V$  as follows. For each directed edge  $(u, v)$  in  $D_U$ , label  $(u, v)$  with the set of vertices that are not equal to  $v$  and not directly before or after  $v$  in the undirected radial ordering around  $u$ . For example, if  $U(u) = v_1, v_2, v_3, v_4$  with  $v = v_2$ , then label  $(u, v)$  with  $\{v_4\}$ . Next, we construct a coloring of the complete undirected graph  $G_U$  on  $V$  by coloring each edge  $\{u, v\}$  green if  $(u, v)$  and  $(v, u)$  have the same label in  $D_U$  and red otherwise. We call  $G_U$  the signature graph of  $U$ . Figure 4 shows several examples.

**Lemma 1.** *Consider an abstract labeled order type  $\chi$  on a set  $V$  with  $|V| = 5$  and let  $U \sim R_\chi$ .*

- (i) *The abstract labeled order types in  $T(U)$  all have the same convex hull size and this size can be computed from  $U$  in constant time.*
- (ii) *If  $\chi$  has convex hull size 4 or 5 then  $T(U) = \{\chi\}$  and  $\chi$  can be computed from  $U$  in constant time.*

*Proof.* Figure 4(a-c) shows the signature graphs of the undirected radial systems of each of the three order type isomorphism classes on five elements. Note that the number of green edges is different for each isomorphism class. This proves (i). For (ii), recall that we want to recover the labeled order type, not just its equivalence class. We perform a case distinction on the isomorphism class of  $\chi$  (which we identify by the number of vertices on the convex hull of  $\chi$ ).



**Fig. 4.** Green edges are solid and red edges are dashed. (a-c) The undirected radial systems of each of the three order type isomorphism classes on five elements. (d-e) The two labeled order types of size five with four vertices on the convex hull, where vertex  $a$  has no incident green edges and  $b$  and  $c$  have one incident green edge.

Suppose that there are five vertices on the convex hull of  $\chi$ . An edge  $\{u, v\}$  is green if and only if  $\{u, v\}$  is on the convex hull. We assume without loss of generality that  $\{a, b\}$  is green. There are six labeled order types of size five with five vertices on the convex hull, under the assumption that  $\{a, b\}$  is on the convex hull. Those order types correspond to sequences starting with  $a, b$  and ending with all six permutations of the three remaining points. The green neighbors of  $a$  and  $b$  thus completely identify the labeled order type.

Suppose now that there are four vertices on the convex hull of  $\chi$ . Referring again to Figure 4(b), we see that there is one vertex with no incident green edges, two vertices with one incident green edge and two vertices with two incident green edges. Without loss of generality, we may assume that the vertex with no incident green edges is vertex  $a$  and the vertices with one incident green edge are  $b$  and  $c$ . This leaves the two labeled order types shown in Figure 4(d-e), which are easily distinguished by the green neighbor of vertex  $b$ .  $\square$

Figure 2(a)-2(b) show that (ii) does not always hold for triangular convex hulls.

### 3 Reconstruction Algorithms

In this section we develop an algorithm to compute  $T(U)$  from an undirected rotation system  $U \sim R_\chi$ . The general approach is the following. We first show, in two steps, that the convex hull  $H$  of  $\chi$  and  $U$  together uniquely determine  $\chi$ . Then we repeatedly apply Lemma 1 to compute  $|H|$  from  $U$ . We show that  $U$  uniquely determines  $H$  if  $|H| \geq 4$ . In that case, we can compute  $T(U) = \{\chi\}$  from  $U$ . Otherwise, if  $|H| = 3$ , we compute  $T(U)$  by trying each possible convex hull. Given an order type  $\chi$  on the vertex set  $V$ , let  $\chi[V']$  be the restriction of  $\chi$  to  $V' \subseteq V$ . We define  $U[V']$  analogously for an undirected radial system  $U$ .

**Lemma 2.** *Consider an abstract labeled order type  $\chi$  on a set  $V$  and let  $U \sim R_\chi$ . Let  $H \subseteq V$  be the set of vertices on the convex hull of  $\chi$ . The pair  $(H, U)$  uniquely determines the cyclic order  $h_1, \dots, h_k$  of the vertices on the convex hull and the clockwise radial system  $R_\chi$  (up to complete reversal of both). Furthermore, there is a polynomial-time algorithm that takes  $(H, U)$  as input and returns  $h_1, \dots, h_k$  and  $R_\chi$ .*



*Proof.* We first give an algorithm to recover the sequence  $h_1, \dots, h_k$ . If  $|H| = 3$  then any ordering of  $H$  will do. If  $4 \leq |H| \leq 5$  then choose any  $H \subseteq V_5 \subset V$  with  $|V_5| = 5$  and use Lemma 1 with  $V_5$  to recover the order type of  $H$  in polynomial time. If  $|H| > 5$ , then let  $h_1, \dots, h_k$  be a cyclic order of  $H$  and consider the signature graph  $G_{U[H]}$ . Note that we can compute the signature graph in polynomial time using only  $U[H]$ . In the digraph  $D_{U[H]}$ , the edges  $(h_i, h_{i+1})$  and  $(h_{i+1}, h_i)$  will both be labeled  $H \setminus \{h_{i-1}, h_i, h_{i+1}, h_{i+2}\}$  and thus  $\{h_i, h_{i+1}\}$  is green in  $G_{U[H]}$  for all  $1 \leq i \leq k$ . On the other hand, the edge  $(h_i, h_j)$  will be labeled  $H \setminus \{h_i, h_{j-1}, h_j, h_{j+1}\}$ , whereas  $(h_j, h_i)$  will be labeled  $H \setminus \{h_{i-1}, h_i, h_{i+1}, h_j\}$  for  $|i-j| > 1$ . Hence,  $\{h_i, h_j\}$  is red in  $G_u$  for all remaining edges. It follows that the green edges in  $G_{U[H]}$  form a hamiltonian cycle which reveals the order of the vertices of  $H$  along the convex hull.

To recover  $R_\chi$ , we assume that  $h_1, \dots, h_k$  is the counterclockwise order and recover the corresponding clockwise radial system  $R_\chi$  (recall that we defined  $\chi = -\chi$ ). For  $|H| \geq 4$ , every  $U(v)$  contains at least three vertices from the convex hull, and hence we can recover the clockwise direction by setting  $R_\chi(v)$  to  $U(v)$  if  $h_1, \dots, h_k$  (without  $v$  if  $v$  is on the convex hull) appear in this order in  $U(v)$  and setting  $R_\chi(v)$  to the reverse of  $U(v)$  otherwise. For  $|H| = 3$  the same procedure works except when  $v$  is on the convex hull. If  $v = h_1$  then the two possible directions are of the form  $h_2, h_3, v_1, v_2, \dots$  and  $h_2, v_1, v_2, \dots, h_3$ . The second one is the correct clockwise order and is easy to recognize (note that if  $V = H$  both orders are identical). The cases  $v = h_2$  and  $v = h_3$  are analogous. This procedure takes polynomial time.  $\square$

We omit the proof of the following lemma due to space limitations; it is essentially an application of Lemma 2, followed by some case analysis to recover  $\chi$ .

**Lemma 3.** *Consider an abstract labeled order type  $\chi$  on a set  $V$  with  $|V| \geq 5$  and let  $U \sim R_\chi$ . Let  $H \subseteq V$  be the set of vertices on the convex hull of  $\chi$ . Then the pair  $(H, U)$  uniquely determines  $\chi$ , i.e.,  $\{\chi' \in T(U) \mid \chi' \text{ has convex hull } H\} = \{\chi\}$ . Furthermore, there is a polynomial-time algorithm that takes  $(H, U)$  as input and returns  $\chi$ .*

**Theorem 1.** *Consider an abstract labeled order type  $\chi$  on a set  $V$  with  $|V| \geq 5$  and let  $U \sim R_\chi$ . There is a polynomial-time algorithm that takes  $U$  as input and returns  $T(U)$ . Furthermore, let  $H$  be the vertices of the convex hull of  $\chi$ . Then*

- (i) *all elements of  $T(U)$  have convex hull size  $|H|$ ; and*
- (ii) *if  $|H| \geq 4$ , then  $T(U) = \{\chi\}$ .*

*Proof.* The algorithm begins by computing a set  $V' \subseteq V$  that contains (at least) all vertices that appear on the convex hull of an order type in  $T(U)$ . Initially, let  $V' := V$ . For each subset  $V_5 \subseteq V$  with  $|V_5| = 5$ , we do the following. By Lemma 1, the elements of  $T(U[V_5])$  all have the same convex hull size  $s$ , and we can compute  $s$  from  $U$  in constant time. If  $s \neq 4$ , we do nothing. If  $s = 4$ , then the algorithm from Lemma 1 in addition returns  $\chi[V_5]$ , and there must be some vertex  $v \in V_5$  that is not on the convex hull of  $\chi[V_5]$ . Note that  $v$  is not on

the convex hull of any order type in  $T(U)$  either. Hence, we delete  $v$  from  $V'$ . After running this procedure for all subsets  $V_5 \subseteq V$  of size 5, we are left with a  $V' \subseteq V$  that contains (at least) all vertices of the convex hulls of all order types in  $T(U)$ . Every 5-element subset of  $V'$  has convex hull size 3 or 5.

We perform a case analysis depending on the size of the set  $V'$ . First suppose that  $|V'| \leq 5$ . If necessary, add back previously deleted vertices to  $V'$  until  $|V'| = 5$ . Use the algorithm from Lemma 1 to recover  $|H|$  from  $V'$ . If  $|H| = 3$ , then continue with the procedure described in the paragraph at the end of this proof. If  $|H| = 4$  or  $|H| = 5$ , then Lemma 1 in addition returns  $\chi[V']$  and thereby  $H$ . Then, by Lemma 3,  $T(U) = \{\chi\}$  and we can compute  $T(U)$  in polynomial time. This shows that (i) and (ii) hold in that case.

Now suppose that  $|V'| > 5$  and note that this implies  $|H| \neq 4$ . If  $|H| = 3$ , then there is a  $V_5 \subset V$  with convex hull size 3. If  $|H| \geq 5$ , then we claim that  $H = V'$ . For the sake of obtaining a contradiction, suppose that there exists a vertex  $v \in V'$  that is not in  $H$ . Fix any triangulation of  $\chi[H]$ . Let  $h_i h_j h_k$  be the cell of the triangulation that contains  $v$  and let  $h_\ell$  be any other vertex of  $H$ . Then  $V_5 = \{h_i, h_j, h_k, h_\ell, v\}$  is a set of five vertices with convex hull size four and  $V_5 \subseteq V'$ , which is a contradiction. We conclude that if  $|H| \geq 5$  then  $H = V'$  and in particular, every  $V_5 \subset V'$  is in convex position. Our algorithm proceeds as follows. If there is a  $V_5 \subset V'$  with convex hull size 3, then we conclude  $|H| = 3$  and continue with the procedure described in the last paragraph below. Otherwise, we conclude that  $H = V'$ . Then  $T(U) = \{\chi\}$  by Lemma 3 and we can compute  $T(U)$  in polynomial time. This finishes the proof of (ii).

It remains to consider the case where the algorithm has established  $|H| = 3$ . If some order type in  $T(U)$  would have convex hull size larger than 3, then the algorithm would already have terminated by the discussion above. Hence, all order types in  $T(U)$  have convex hull size 3, which completes the proof of (i).

Finally, we describe what the algorithm does when  $|H| = 3$ . Consider all subsets  $H_3 \subseteq V$  of size 3. For each such  $H_3$ , run the algorithm from Lemma 3 with  $(H_3, U)$ , which returns a function  $\chi$ . If  $H_3$  is the convex hull of an order type in  $T(U)$  then  $\chi \in T(U)$  and  $\chi$  is the only order type in  $T(U)$  with convex hull  $H_3$ . If no order type in  $T(U)$  has convex hull  $H_3$ , then the output  $\chi$  is undefined. Hence, it is sufficient to check for each  $H_3$  whether  $\chi$  is an order type (in polynomial time, using the order type axioms) and if so, whether  $U \sim R_\chi$ . If and only if both conditions hold, then  $\chi \in T(U)$  and hence  $T(U)$  can be computed. Since there are  $O(|V|^3)$  subsets of size 3 in  $V$ , the algorithm runs in polynomial time.  $\square$

Given a set  $V$  and for each  $v \in V$  a permutation of  $V \setminus \{v\}$ , we can decide in polynomial time whether this is a radial system corresponding to an actual order type. This is done by running the algorithm above until either an inconsistency is detected or an output is produced. If one of the chirotopes in the output has radial system  $U$  then the answer to the decision problem is yes, and no otherwise.

**Corollary 1.** *Given a set  $V$  and for each  $v \in V$  a permutation of  $V \setminus \{v\}$ , we can decide in polynomial time whether this is the radial system of some order type.*

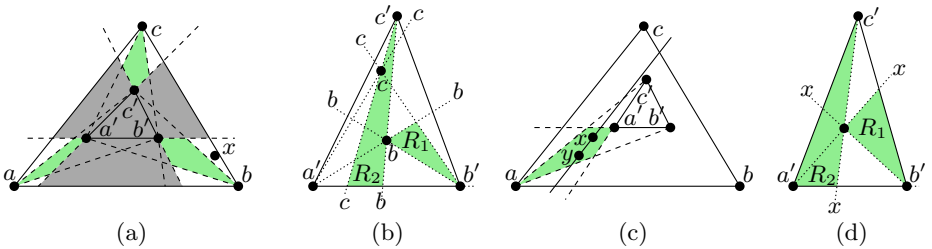
## 4 Triangular Convex Hulls

Theorem 1 only guarantees the trivial bound  $|T(U)| \leq |V|^3$  when a radial system  $U$  has a triangular convex hull. As discussed in the introduction, there are radial systems  $U$  with  $|T(U)| \geq |V| - 1$ . We next prove the matching upper bound.

Recall that  $U$  and the convex hull together uniquely determine the labeled order type (Lemma 3). We also know that if  $U$  is the undirected radial system of a labeled order type with a triangular convex hull, then all order types in  $T(U)$  have a triangular convex hull (Theorem 1). If a triangle  $a, b, c \in V$  is the convex hull for some order type in  $T(U)$ , we say that  $abc$  is *important* (with respect to  $U$ ). Note that if  $abc$  is important, then  $b$  and  $c$  must appear consecutively in the radial ordering of  $a$  (and the analogous statements for  $b$  and  $c$  also hold). We capture the relations between important triangles with the following four propositions. In each proposition, we consider an abstract labeled order type  $\chi$  on a set  $V$  with  $|V| \geq 5$  and a triangular convex hull and a  $U \sim R_\chi$ .

**Proposition 1.**  *$U$  has at most two disjoint important triangles. If  $U$  has exactly two disjoint important triangles, then these are the only important triangles and hence  $|T(U)| \leq 2$ .*

*Proof.* Suppose that  $U$  has disjoint important triangles  $abc$  and  $a'b'c'$ . We now argue that without loss of generality,  $c', a', b'$  appear consecutively and in this order in  $U(a)$ . Figure 5(a) depicts the order type where  $abc$  forms the convex hull. Since  $b'$  and  $c'$  must appear consecutively in  $U(a')$  and since  $a'$  is not on the convex hull, the cone  $b'a'c'$  must not contain any other vertices. The same argumentation for  $b'$  and  $c'$  shows that the dark gray region in Figure 5(a) must be empty. We wish to show that all remaining vertices must be in the light green regions. So suppose there is a vertex  $x$  outside both the dark gray and light green regions. By symmetry we may assume that it is in the position indicated by Figure 5(a). In the order type where  $a'b'c'$  forms the convex hull,  $U(a')$  and  $U(b')$  force  $x$  to be in region  $R_1$  in Figure 5(b). But  $U(c')$  forces  $x$  to be in region  $R_2$ , which is disjoint from  $R_1$  (except vertex  $b$ ). Hence,  $a'b'c'$  cannot form the convex hull, which is a contradiction. We conclude that all remaining vertices must be in the light green regions in Figure 5(a). We call the complement of the light green regions the *forbidden region* of  $a'b'c'$ .



**Fig. 5.** Two disjoint important triangles. (a-b) Vertex  $x$  cannot be in the indicated position. (c-d) The supporting line of  $xy$  cannot avoid the segment  $b'c'$ .

We claim that the supporting line  $\overline{xy}$  of two vertices  $x$  and  $y$  in a light green region in Figure 5(a) must separate the other two green regions. Note that this holds trivially if one of  $x$  and  $y$  is  $a$  or  $a'$ . Otherwise, suppose without loss of generality that  $x$  and  $y$  are in the light green region incident to  $a$ , that a clockwise sweep from  $c$  to  $b$  around  $a$  encounters  $x$  before  $y$ , that  $\overline{xy}$  does not intersect  $c'b'$  and that  $c'$  is below  $\overline{xy}$ . See Figure 5(c). Looking at the order type where  $a'b'c'$  forms the outer face (Figure 5(d)), we see that  $U(a')$  and  $U(b)$  force  $y$  to be in region  $R_1$ . But  $U(c')$  forces  $y$  to be in region  $R_2$ , which is a contradiction. Hence, the supporting line of  $x$  and  $y$  in Figure 5(a) must intersect  $c'$  and  $b'$  and thus separate the light green regions incident to  $b$  and  $c$ .

Finally, we argue that there are no other important triangles. Consider again the order type depicted in Figure 5(a). Suppose that there is another important triangle  $\Delta$ . Suppose that  $\Delta$  is completely inside one light green region, say the one incident to  $a$ . Since all three supporting lines of  $\Delta$  separate the other two light green regions, either  $b$  or  $c$  must be in the forbidden region of  $\Delta$ , which is a contradiction. Similarly, if  $\Delta$  has one vertex in every light green region, then at least one of  $a'$ ,  $b'$  and  $c'$  is strictly inside  $\Delta$  and hence in  $\Delta$ 's forbidden region. Hence,  $\Delta$  must have two vertices  $a''$  and  $b''$  in one light green region, say the one incident to  $a$ , and one vertex  $c''$  in another light green region, say the one incident to  $c$ . We must have  $c'' = c$ : otherwise  $c$  is in the forbidden region of  $\Delta$ . But then  $c'$  is in the forbidden region of  $\Delta$ , which is a contradiction. Hence, there are only two important triangles and thus  $|T(U)| \leq 2$  by Lemma 3.  $\square$

**Proposition 2.** *If there is a vertex  $v^*$  that is common to all important triangles in  $U$ , then  $|T(U)| \leq |V| - 1$ .*

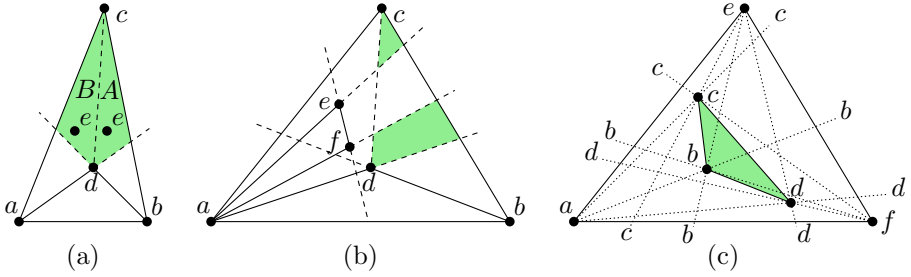
*Proof.* For every important triangle  $v^*uw$  we know that  $u$  and  $w$  must be consecutive in  $U(v^*)$ . Since there are only  $|V| - 1$  consecutive pairs in  $U(v^*)$ , the proposition follows immediately by Lemma 3.  $\square$

We omit the proof of the following proposition due to space limitations; it is similar to the proof of Proposition 1.

**Proposition 3.** *If every pair of important triangles has exactly one vertex in common, then all important triangles must all have the same vertex in common.*

**Proposition 4.** *If there exists a pair of important triangles with two vertices in common, then all important triangles must have the same vertex in common.*

*Proof.* Let  $abc$  and  $abd$  be the important triangles from the statement. Suppose for the sake of obtaining a contradiction that not all important triangles share the same vertex, i.e., that there is an important triangle  $\Delta_1$  that does not contain  $a$  and an important triangle  $\Delta_2$  that does not contain  $b$ , with possibly  $\Delta_1 = \Delta_2$ . If  $\Delta := \Delta_1 = \Delta_2$ , then by Proposition 1 we have  $\Delta = cde$  with  $e \neq a, b$ . See Figure 6(a). The forbidden region of  $\Delta$  contains  $a$  if  $e$  is in the light green region  $A$  and it contains  $b$  otherwise. It follows that  $\Delta_1 \neq \Delta_2$ .



**Fig. 6.** Two important triangles that share two vertices. (a) Triangle  $cde$  cannot be important. (b) Possible locations for the vertex  $x$ . (c) Contradiction to (b) when  $ae f$  forms the convex hull.

Since  $\Delta_1$  is not disjoint from  $abc$  and  $abd$  by Proposition 1 (since we have four important triangles),  $\Delta_1$  must contain  $b$  or both  $c$  and  $d$ . Similarly,  $\Delta_2$  must contain  $a$  or both  $c$  and  $d$ . Suppose that  $\Delta_1$  contains both  $c$  and  $d$  and let  $e$  be the third vertex of  $\Delta_1$ . By the argument in the previous paragraph, we must have  $e = b$ . But then the forbidden regions of  $abd$  and  $\Delta_1 = bcd$  together cover all of  $abc$ . This is a contradiction since  $|V| \geq 5$ . Symmetrically,  $\Delta_2$  cannot contain both  $c$  and  $d$ . Hence,  $\Delta_1$  must contain  $b$  and  $\Delta_2$  must contain  $a$ . Furthermore, neither triangle can intersect  $cd$  since  $c$  or  $d$  would be in the forbidden region otherwise. Let  $\Delta_2 = aef$  such that a clockwise sweep from  $c$  to  $d$  around  $a$  encounters  $e$  and  $f$  in this order (with possibly  $e = c$  or  $f = d$  but not both). Let  $x$  be a vertex of  $\Delta_1$  different from  $b, c$  and  $d$ . The light green region in Figure 6(b) shows the allowed locations for  $x$ . The supporting line of  $ef$  cannot intersect  $cd$  since  $c$  or  $d$  would be in the forbidden region of  $\Delta_2$  otherwise. Figure 6(c) shows the resulting order type where  $ae f$  forms the convex hull. The radial orderings of  $b, c$  and  $d$  force  $x$  to be in the light green region. Referring to Figure 6(b), we see that  $d, b$  and  $c$  appear consecutively in  $U(x)$ . But in Figure 6(c), this certainly cannot be the case, even if  $c = e$  or  $d = f$ , which contradicts our assumption. We conclude that we cannot have such  $\Delta_1$  and  $\Delta_2$  and therefore that all important triangles must share a vertex.  $\square$

It now follows from Propositions 1, 2, 3, and 4 that:

**Theorem 2.** Consider an abstract labeled order type  $\chi$  on a set  $V$  with  $|V| \geq 5$  and let  $U \sim R_\chi$ . Then  $|T(U)| \leq |V| - 1$ .

## 5 Discussion and Open Problems

Theorem 2 cannot be improved by considering clockwise radial systems instead of undirected ones. For  $|H| \geq 4$ , the undirected radial system is already sufficient to reconstruct the order type. For  $|H| = 3$ , the worst case example from Figure 2(a)-2(b) applies even for clockwise radial systems.

In terms of future work, an axiomatic characterization of radial systems could lead to a simpler recognition algorithm. Our algorithms are obtained as

byproducts of the proofs and their running time can undoubtedly be improved. Finally, one could think of generalizing the problem to higher dimensions. Instead of a cyclic ordering of points, every point  $p$  of a set in  $\mathbb{R}^3$  could be associated with a rank-3 oriented matroid obtained by projecting all other points on a small sphere around  $p$ . The higher-dimensional counterparts of local sequences were defined for instance by Bokowski et al. [4] and are called *hyperline sequences*.

**Acknowledgments.** This work was initiated at the Joint EuroGIGA ComPoSe-VORONOI Meeting in Graz (Austria) on July 8-12, 2013. It was pursued at the ComPoSe Workshop on Algorithms using the Point Set Order Type in Ratsch (Austria) on March 31-April 1, 2014. We wish to thank the organizers of these two meetings as well as the other participants. We would like to thank Alexander Pilz in particular for helpful discussions on this topic.

## References

1. Goodman, J.E., Pollack, R.: Multidimensional sorting. *SIAM Journal on Computing* **12**(3), 484–507 (1983)
2. Knuth, D.E. (ed.): *Axioms and Hulls*. LNCS, vol. 606. Springer, Heidelberg (1992)
3. Björner, A., Las Vergnas, M., Sturmfels, B., White, N., Ziegler, G.: *Oriented Matroids*, 2nd edn. Cambridge University Press (1999)
4. Bokowski, J., King, S., Mock, S., Streinu, I.: The topological representation of oriented matroids. *Discrete & Computational Geometry* **33**(4), 645–668 (2005)
5. Goodman, J.E., Pollack, R.: Semispaces of configurations, cell complexes of arrangements. *J. Comb. Theory, Series A* **37**(3), 257–293 (1984)
6. Felsner, S.: On the number of arrangements of pseudolines. In: *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pp. 30–37. ACM (1996)
7. Felsner, S., Valtr, P.: Coding and counting arrangements of pseudolines. *Discr. & Comp. Geom.* **46**(3), 405–416 (2011)
8. Pilz, A., Welzl, E.: Order on order-types. In preparation
9. Tovar, B., Freda, L., LaValle, S.M.: Using a robot to learn geometric information from permutations of landmarks. *Contemporary Mathematics* **438**, 33–45 (2007)
10. Wismath, S.K.: Point and line segment reconstruction from visibility information. *Int. J. Comput. Geometry Appl.* **10**(2), 189–200 (2000)
11. Dissler, Y., Mihalák, M., Widmayer, P.: Reconstructing a simple polygon from its angles. In: Kaplan, H. (ed.) *SWAT 2010*. LNCS, vol. 6139, pp. 13–24. Springer, Heidelberg (2010)
12. Chen, D.Z., Wang, H.: An improved algorithm for reconstructing a simple polygon from its visibility angles. *Comput. Geom.* **45**(5–6), 254–257 (2012)
13. Devillers, O., Dujmovic, V., Everett, H., Hornus, S., Whitesides, S., Wismath, S.K.: Maintaining visibility information of planar point sets with a moving viewpoint. *Int. J. Comput. Geometry Appl.* **17**(4), 297–304 (2007)
14. Díaz-Báñez, J.M., Fabila-Monroy, R., Pérez-Lantero, P.: On the number of radial orderings of colored planar point sets. In: Márquez, A., Ramos, P., Urrutia, J. (eds.) *EGC 2011*. LNCS, vol. 7579, pp. 109–118. Springer, Heidelberg (2012)
15. Durocher, S., Mehrabi, S., Mondal, D., Skala, M.: Realizing site permutations. In: *CCCG*. (2011)
16. Cardinal, J., Kusters, V.: The complexity of simultaneous geometric graph embedding. *CoRR abs/1302.7127* (2013)

# A Randomized Divide and Conquer Algorithm for Higher-Order Abstract Voronoi Diagrams

Cecilia Bohler<sup>1</sup>, Chih-Hung Liu<sup>1</sup>,  
Evanthia Papadopoulou<sup>2</sup>(✉), and Maksym Zavershynskiy<sup>2</sup>

<sup>1</sup> Institute of Computer Science I, University of Bonn, 53113 Bonn, Germany  
bohler@cs.uni-bonn.de, chliu@uni-bonn.de

<sup>2</sup> Faculty of Informatics, Università della Svizzera italiana (USI),  
Lugano, Switzerland  
{evanthia.papadopoulou,maksym.zavershynskiy}@usi.ch

**Abstract.** Given a set of sites in the plane, their order- $k$  Voronoi diagram partitions the plane into regions such that all points within one region have the same  $k$  nearest sites. The order- $k$  abstract Voronoi diagram is defined in terms of bisecting curves satisfying some simple combinatorial properties, rather than the geometric notions of sites and distance, and it represents a wide class of order- $k$  concrete Voronoi diagrams. In this paper we develop a randomized divide-and-conquer algorithm to compute the order- $k$  abstract Voronoi diagram in expected  $O(kn^{1+\varepsilon})$  operations. For solving small sub-instances in the divide-and-conquer process, we also give two sub-algorithms with expected  $O(k^2n \log n)$  and  $O(n^2 2^{\alpha(n)} \log n)$  time, respectively. This directly implies an  $O(kn^{1+\varepsilon})$ -time algorithm for several concrete order- $k$  instances such as points in any convex distance, disjoint line segments and convex polygons of constant size in the  $L_p$  norm, and others.

**Keywords:** Higher-Order Voronoi Diagram · Abstract Voronoi Diagram · Randomized Algorithm · Divide and Conquer

## 1 Introduction

Given a set  $S$  of  $n$  geometric sites in the plane, their order- $k$  *Voronoi diagram*,  $V_k(S)$ , is a subdivision of the plane such that every point within an order- $k$  *Voronoi region* has the same  $k$  nearest sites. The common boundary between two adjacent Voronoi regions is a *Voronoi edge*, and the common vertex incident to more than two Voronoi regions is a *Voronoi vertex*. The ordinary Voronoi diagram is the order-1 Voronoi diagram, and the farthest-site Voronoi diagram is the order- $(n-1)$  Voronoi diagram.

---

This work was supported by the European Science Foundation (ESF) in the EURO-CORES collaborative research project EuroGIGA/VORONOI, projects DFG KI 655/17-1 and SNF 20GG21-134355. The work of the last two authors was also supported by the Swiss National Science Foundation, project 200020-149658.

For point sites in the Euclidean metric, the order- $k$  Voronoi diagram has been well-studied. Lee [14] showed its structural complexity to be  $O(k(n-k))$ , and proposed an  $O(k^2n \log n)$ -time iterative algorithm. Based on the notions of arrangements and geometric duality, Chazelle and Edelsbrunner [6] developed an algorithm with  $O(n^2+k(n-k) \log^2 n)$  time complexity. Clarkson [7] developed an  $O(kn^{1+\varepsilon})$ -time randomized divide-and-conquer algorithm, and Agarwal et al. [1], Chan [5], and Ramos [18] proposed randomized incremental algorithms with  $O(k(n-k) \log n + n \log^3 n)$ ,  $O(n \log n + nk \log k)$ , and  $O(n \log n + nk 2^{O(\log^+ k)})$  time complexities, respectively. Besides, Boissonnat et al. [4] and Aurenhammer and Schwarzkopf [2] also studied on-line algorithms.

Surprisingly, order- $k$  Voronoi diagrams of sites other than points were only recently considered [17] illustrating different properties from their counterparts for points. For simple, even disjoint, line segments, a single order- $k$  Voronoi region may consist of  $\Omega(n)$  disjoint faces; nevertheless, the overall structural complexity of the diagram for  $n$  non-crossing line segments remains  $O(k(n-k))$  [17]. Abstract Voronoi diagrams were introduced by Klein [10] as a unifying concept to many instances of concrete Voronoi diagrams. They are defined in terms of a system of bisecting curves  $\mathcal{J} = \{J(p, q) \mid p, q \in S, p \neq q\}$  rather than concrete geometric sites and distance measures. Order- $k$  abstract Voronoi diagrams were recently considered in [3], providing a unified concept to order- $k$  Voronoi diagrams, and showing the number of their faces to be  $\leq 2k(n-k)$ . No algorithms for their construction have been available so far. For non-point sites, such as line segments, only  $O(k^2n \log n)$ -time algorithms have been available based on the iterative construction [17] and plane sweep [19]. Other recent works on order- $k$  Voronoi diagrams of point-sites in generalized metrics include the  $L_1/L_\infty$  metric [16], the city metric [8], and the geodesic order- $k$  Voronoi diagram [15].

In abstract Voronoi diagrams [10], the system of bisecting curves satisfies axioms (A1)–(A5), given below, for any  $S' \subseteq S$ . Once a concrete bisector system is shown to satisfy these axioms, combinatorial properties and algorithms to construct abstract Voronoi diagrams (see e.g., [10]) are directly applicable. A bisector  $J(p, q)$  partitions the plane into two domains  $D(p, q)$  and  $D(q, p)$ , where  $D(p, q)$  are points closer to  $p$  than  $q$ ; a first-order Voronoi region  $\text{VR}_1(\{p\}, S)$  is defined as  $\bigcap_{q \in S, q \neq p} D(p, q)$ .

- (A1). Each first-order Voronoi region is pathwise connected.
- (A2). Each point in the plane belongs to the closure of some first-order Voronoi region.
- (A3). No first-order Voronoi region is empty.
- (A4). Each curve  $J(p, q)$ , where  $p \neq q$ , is unbounded. After stereographic projection to the sphere, it can be completed to be a closed Jordan curve through the north pole.
- (A5). Any two curves  $J(p, q)$  and  $J(s, t)$  have only finitely many intersection points, and these intersections are transversal.



In this paper, we develop a randomized divide-and-conquer algorithm to compute the order- $k$  abstract Voronoi diagram in expected  $O(kn^{1+\varepsilon})$  basic operations, based on Clarkson’s random sampling technique and one additional axiom:

**(A6).** The number of vertical tangencies of a bisector is  $O(1)$ .

Our algorithm is applicable to a variety of concrete order- $k$  Voronoi diagrams satisfying axioms (A1)-(A6), such as point sites in any convex distance metric or the Karlsruhe metric, disjoint line segments and disjoint convex polygons of constant size in the  $L_p$  norms, or under the Hausdorff metric. In these instances, all basic operations (see Section 2) can be performed in  $O(1)$  time, thus, our algorithm runs in expected  $O(kn^{1+\varepsilon})$  time. For non-point sites, this is the first algorithm that achieves time complexity different from the standard  $O(k^2n \log n)$ , which is efficient for only small values of  $k$ . For point sites in the Euclidean metric, near-optimal randomized algorithms exist [1],[5],[7],[18]; however, they are based on powerful geometric transformations, which are non-trivial to convert to different geometric objects, and/or to the abstract setting, which is based on topological (non-geometric) properties. Matching the time complexity of these algorithms in the abstract setting or for concrete non-point instances remains an open problem.

In order to apply Clarkson’s technique [7], we define a vertical decomposition of the order- $k$  Voronoi diagram. We prove that our vertical trapezoidal decomposition allows a divide-and-conquer algorithm and an expected time analysis. When the problem sub-instances are small enough, we propose two sub-algorithms. The first one combines the standard iterative approach [14] and the randomized incremental construction for the order-1 abstract Voronoi diagram [12] and computes the order- $k$  abstract Voronoi diagram in expected  $O(k^2n \log n)$  operations. For the second one, we adopt Har-Peled’s method [9] and obtain an  $O(n^2 2^{\alpha(n)} \log n)$ -operation randomized algorithm, where  $\alpha(\cdot)$  is the inverse of the Ackermann function. Our algorithm follows the essence of Clarkson’s randomized divide-and-conquer algorithm for the Euclidean order- $k$  Voronoi diagram [7], however, it bypasses all geometric transformations and constraints. Instead, our algorithm defines sub-structures and conflict relations relying on the properties of a bisector system that satisfies the six axioms (A1)–(A6).

## 2 Preliminaries

Axioms (A1)-(A5) imply that for a given bisecting system  $\mathcal{J}$  and a fixed point  $x \in \mathbb{R}^2$  we can define a linear order on the sites in  $S$ .

**Definition 1.** For a point  $x \in \mathbb{R}^2$  and two sites  $p, q \in S$ ,  $p <_x q$ ,  $p =_x q$ , or  $p >_x q$  if  $x \in D(p, q)$ ,  $x \in J(p, q)$ , or  $x \in D(q, p)$ , respectively.

Since  $D(p, q) \cap D(q, r) \subseteq D(p, r)$  [10, 11], we can define an ordered sequence on  $S$ ,  $\pi_x^S = (s_1, \dots, s_n)$ , given  $x$ , satisfying  $s_1 \leq_x s_2 \leq_x \dots \leq_x s_n$ . We say that site  $s$  is  $k$ -nearest to point  $x$  if  $s$  occupies the  $k$ -th position in the sequence  $\pi_x^S$ .

**Definition 2.** [3] *The order- $k$  Voronoi region associated with  $H$  is*

$$VR_k(H, S) = \bigcap_{p \in H, q \in S \setminus H} D(p, q).$$

*The order- $k$  Voronoi diagram is*

$$V_k(S) = \bigcup_{|H|=k} \partial VR_k(H, S),$$

*where  $\partial$  denotes the boundary.*

For each point  $x \in VR_k(H, S)$  and  $\pi_x^S = (s_1, \dots, s_n)$ ,  $H = \{s_1, \dots, s_k\}$ , and  $s_k <_x s_{k+1}$ . If  $VR_k(H_1, S)$  and  $VR_k(H_2, S)$  share an edge  $e$ , then for any point  $x \in e$ ,  $H_1 \cap H_2 = \{s_1, \dots, s_{k-1}\}$  and  $s_{k-1} <_x s_k =_x s_{k+1}$ , see [3, Lemma 5]. For simplicity, throughout this paper, we make a general position assumption that the degree of any Voronoi vertex is exactly three.

**Definition 3.** *Let  $v$  be a Voronoi vertex among  $VR_k(H_1, S)$ ,  $VR_k(H_2, S)$ , and  $VR_k(H_3, S)$ , and let  $H = H_1 \cap H_2 \cap H_3$  then  $v$  can be categorized into two types: new when  $|H| = k - 1$  and old when  $|H| = k - 2$ .*

A new Voronoi vertex of  $V_k(S)$  is an old Voronoi vertex of  $V_{k+1}(S)$ .

Let  $v$  be a Voronoi vertex as in Def. 3. Then we can show that  $H = \{s_1, \dots, s_t\}$  and  $s_t <_v s_{t+1} =_v s_{t+2} =_v s_{t+3} <_v s_{t+4}$ , where  $t = |H|$  and  $\pi_v^S = (s_1, \dots, s_n)$ . Each Voronoi vertex is defined by the three sites  $s_{t+1}, s_{t+2}, s_{t+3}$ .

**Definition 4.** *The  $k$ -neighborhood of a site  $p$  in  $S$ , denoted by  $VN_k(p, S)$ , is the union of closures of  $VR_k(H, S)$  for all  $H \subset S$ , such that  $p \in H$  and  $|H| = k$ , i.e.,*

$$VN_k(p, S) = \bigcup_{p \in H, H \subset S, |H|=k} \overline{VR_k(H, S)},$$

*where  $\overline{X}$  denotes the topological closure of the set  $X$ .*

Each edge of  $\partial VN_k(p, S)$  belongs to  $J(p, q)$  for a site  $q \in S \setminus \{p\}$ , and each edge of  $V_k(S)$  belongs to  $\partial VN_k(p, S)$  for a site  $p \in S$ . The latter condition implies

$$V_k(S) = \bigcup_{p \in S} \partial VN_k(p, S).$$

Unlike order- $k$  Voronoi regions of point-sites, abstract order- $k$  Voronoi regions may be disconnected. In fact one region may disconnect into  $\Omega(n)$  disjoint faces, for  $k > 1$  (see e.g. [17] for line segments). Nevertheless, the  $k$ -neighborhood is connected, and this is the major property used in Section 5.

**Lemma 1.**  *$VN_k(p, S)$  is simply connected and there is no finite set of points whose removal would make  $VN_k(p, S)$  disconnected.*

*Proof.* First we show that  $\text{VN}_k(p, S)$  is path connected. The definition of  $\text{VN}_k(p, S)$  implies that  $p$  is at most  $k$ -nearest for every point in  $\text{VN}_k(p, S)$ . Therefore  $\text{VN}_k(p, S) = \bigcup_{p \in H, H \subset S, |H|=k} \overline{\text{VR}_1(p, \{p\} \cup (S \setminus H))}$ .  $\text{VR}_1(p, \{p\} \cup (S \setminus H))$  is path connected, axiom (A1). Thus the connectivity of  $\text{VN}_k(p, S)$  follows.

Next we show that there can be no holes in  $\text{VN}_k(p, S)$ . Suppose there is a face  $F$  entirely surrounded by  $\text{VN}_k(p, S)$ . Then all edges on the boundary of  $F$  are subsets of  $\partial \text{VN}_k(p, S)$ . Let the edges correspond to the bisectors  $J(p, q_i)$ ,  $i = 1, \dots, m$ . If one of the bisectors  $J(p, q_i)$  goes through the interior of  $F$  then consider a face of  $F \cap D(q_i, p)$ , which is not empty, and so on until we have a face  $F'$  bounded by edges  $J(p, q'_1), \dots, J(p, q'_{m'})$  and  $F' \subset D(q'_1, p) \cap \dots \cap D(q'_{m'}, p)$ . This implies that  $F'$  is a bounded face of the farthest Voronoi region of  $p$  in  $\{p, q'_1, \dots, q'_{m'}\}$ , a contradiction [3, Lemma 7].  $\square$

Our algorithm, to be described in the sequel, assumes the availability of the following basic operations. (1) For an arbitrary point  $x$ , determine if  $x$  is in  $D(p, q)$ ,  $J(p, q)$  or  $D(q, p)$ ; (2) Given a point  $x$  on  $J(p, q)$ , determine the next vertical tangent point or the next intersection with  $J(s, t)$  or a straight line along one direction of  $J(p, q)$ ; (3) For two points  $x, y$  on  $J(p, q)$ , determine the in-front/behind relation along one direction of  $J(p, q)$ ; (4) For two points  $x$  and  $y$  compare them by  $x$ -coordinate, where  $x$  and  $y$  are intersection points or points of vertical tangency of the bisectors.

### 3 Randomized Divide and Conquer Algorithm

#### 3.1 Refined Diagram

We first refine  $V_k(S)$  and partition it into vertical trapezoids.

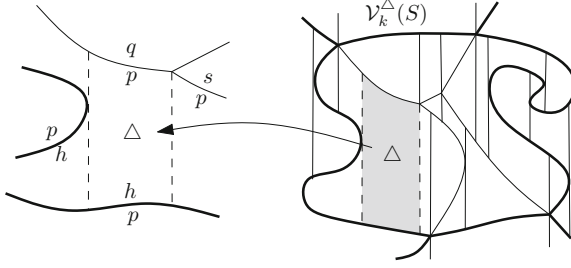
**Definition 5.** *The refined order- $k$  Voronoi diagram  $\mathcal{V}_k(S)$  of  $S$  is derived by superimposing  $V_k(S)$  and  $V_{k+1}(S)$ . It is defined as:*

$$\mathcal{V}_k(S) = V_k(S) \cup \bigcup_{H \subset S, |H|=k} V_1(S \setminus H) \cap \text{VR}_k(H, S).$$

A region  $\mathcal{VR}_k(p, H, S)$  of  $\mathcal{V}_k(S)$  is associated with a site  $p \in S$ , which is called the dominator, and a  $k$ -element subset  $H \subset S$ . For any point  $x \in \mathcal{VR}_k(p, H, S)$ ,  $H$  is the set of  $k$  nearest sites to  $x$  and  $p$  is the  $(k+1)$ -nearest site to  $x$ .

**Definition 6.** *The vertical decomposition of  $\mathcal{V}_k(S)$ , denoted by  $\mathcal{V}_k^\Delta(S)$ , is the subdivision of the plane into (pseudo-)trapezoids obtained by shooting vertical rays up and down from each vertex in  $\mathcal{V}_k(S)$  and each vertical tangent point of each edge in  $\mathcal{V}_k(S)$ , until the intersection with an edge or all the way to infinity.*

**Lemma 2.**  *$\mathcal{V}_k^\Delta(S)$  can be constructed from  $V_k(S)$  in expected  $O(k(n-k) \log n)$  operations.*



**Fig. 1.** Trapezoid  $\Delta$  of  $V_k^\Delta(S)$ .  $V_k(S)$  is depicted in bold.

A trapezoid  $\Delta$  of  $V_k^\Delta(S)$  in  $\mathcal{VR}_k(p, H, S)$  is defined by the dominator  $p$  and 1-4 other sites. Vertical boundaries of the trapezoid may be defined either by an intersection point or by a point of vertical tangency. Moreover, one of the vertical boundaries may be degenerate. Let  $d(\Delta)$  be the dominator of the trapezoid and  $B(\Delta)$  be the set of sites that together with the dominator define the boundaries of the trapezoid  $\Delta$ . Then  $1 \leq |B(\Delta)| \leq 4$  and for any point  $x \in \Delta$ ,  $H \setminus B(\Delta)$  are the  $k - |H \cap B(\Delta)|$  nearest sites to  $x$ .

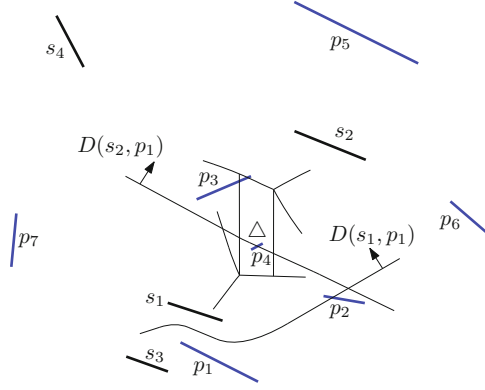
In Fig. 1, the top and bottom edges of  $\Delta$  are defined by  $J(p, q)$  and  $J(p, h)$ , respectively, and the left and right edges are defined by a vertical tangent point of  $J(p, h)$  and an intersection between  $J(p, q)$  and  $J(p, s)$ , respectively. In other words,  $B(\Delta) = \{q, h, s\}$  and  $d(\Delta) = p$ .

**Definition 7.** For a trapezoid  $\Delta$  of  $V_k^\Delta(S)$ , a site  $s \notin B(\Delta)$  strongly conflicts with  $\Delta$ , if  $\overline{\Delta} \subset D(s, d(\Delta))$ . A site  $s \notin B(\Delta)$  weakly conflicts with  $\Delta$ , if  $\overline{\Delta} \cap D(s, d(\Delta)) \neq \emptyset$ . The set of sites  $X \subseteq S$  that strongly, resp. weakly conflict with  $\Delta$  is denoted by  $X \wedge_s \Delta$ , resp.  $X \wedge_w \Delta$ .

In general, the set of *strong conflicts* is different from the set of *weak conflicts*, and  $X \wedge_s \Delta \subseteq X \wedge_w \Delta$ . In Figure 2, set  $S = \{p_1, \dots, p_7, s_1, \dots, s_4\}$  is the set of line segments in Euclidean space.  $R = \{p_1, \dots, p_7\}$  is the subset of  $S$  and  $\Delta$  is the trapezoid of  $V_3^\Delta(R)$  in  $\mathcal{VR}_3(p_1, \{p_2, p_3, p_4\}, R)$ . The dominator  $d(\Delta)$  of the trapezoid  $\Delta$  is  $p_1$ . The set of the sites  $B(\Delta)$  that define the boundaries of the trapezoid  $\Delta$  is  $\{p_2, p_3, p_5, p_6\}$ . Since the sites  $p_2, p_3, p_5, p_6$  define the boundary of the trapezoid they cannot conflict with the trapezoid. However, the site  $p_4$  strongly conflicts with  $\Delta$ , since  $\overline{\Delta} \subset D(p_4, p_1)$ . Sites that do not belong to  $R$  can also conflict with the trapezoid. Here, site  $s_1$  strongly conflicts with  $\Delta$ , since  $\overline{\Delta} \subset D(s_1, p_1)$ . However, site  $s_2$  weakly conflicts with  $\Delta$ , because the dominance region  $D(s_2, p_1)$  does not enclose  $\overline{\Delta}$ , but only intersects  $\overline{\Delta}$ . Thus,  $S \wedge_s \Delta = \{p_4, s_1\}$ ,  $S \wedge_w \Delta = \{p_4, s_1, s_2\}$ . In Lemmata 3, 4 we use *weak* and *strong conflicts* for the upper and lower bounds, respectively.

**Lemma 3.** Let  $R$  be a subset of  $S$  and  $\beta$  be a positive integer. Then for any trapezoid  $\Delta$  of  $V_\beta^\Delta(R)$ ,  $\beta - 4 \leq |R \wedge_s \Delta|$  and  $|R \wedge_w \Delta| \leq \beta$ .

*Proof.* Let  $\Delta$  be in  $\mathcal{VR}_\beta(H, R)$ . We want to prove that  $H \setminus B(\Delta) \subseteq R \wedge_s \Delta$  and  $R \wedge_w \Delta \subseteq H$ . Since for each point  $x \in \Delta$ ,  $H$  are the  $\beta$  nearest sites of  $x$  and



**Fig. 2.** Trapezoid  $\Delta \in \mathcal{VR}_3(p_1, \{p_2, p_3, p_4\})$ , where  $p_1, \dots, p_7$  are line segments

$d(\Delta)$  is the  $(\beta+1)$ -nearest site, for each site  $p \in H \setminus B(\Delta)$ ,  $\overline{\Delta} \subset D(p, d(\Delta))$ , implying that  $H \setminus B(\Delta) \subseteq R \wedge_s \Delta$ . For each site  $p \in R \wedge_w \Delta$ ,  $D(p, d(\Delta))$  must include  $\Delta$ ; otherwise,  $d(\Delta)$  is not the  $(\beta+1)$ -nearest site for all points in  $\Delta$ . By Def. 2,  $p$  must belong to  $H$ , implying that  $R \wedge_w \Delta \subseteq H$ .  $\square$

Lemma 3 and [7, Corollaries 4.3 and 4.4] imply the following.

**Lemma 4.** *Let  $R$  be an  $r$ -element random sample of  $S$ . Then with probability at least  $1/2$ , as  $r \rightarrow \infty$ , for any  $\Delta \in \mathcal{V}_\beta^\Delta(R)$ ,  $|S|/(r-5) \leq |S \wedge_s \Delta|$  and  $|S \wedge_w \Delta| \leq \alpha|S|$ , where  $\beta = O(\log r / \log \log r)$  and  $\alpha = O(\log r / r)$ .*

**Lemma 5.** *Let  $R$  be a subset of  $S$  such that for any trapezoid  $\Delta \in \mathcal{V}_\beta^\Delta(R)$ ,  $|S \wedge_s \Delta| > k$ . Let  $v$  be a Voronoi vertex of  $V_k(S)$ . Then there exists a trapezoid  $\Delta \in \mathcal{V}_\beta^\Delta(R)$  such that  $v$  is also a Voronoi vertex of  $V_k(S \wedge_w \Delta)$ .*

*Proof.* (Sketch) Let  $v$  be a Voronoi vertex incident to Voronoi regions  $\text{VR}_k(H_1, S)$ ,  $\text{VR}_k(H_2, S)$  and  $\text{VR}_k(H_3, S)$ , and let  $\Delta$  be a trapezoid of  $\mathcal{V}_\beta^\Delta(R)$  such that  $v \in \overline{\Delta}$ . We want to prove that  $H_1 \cup H_2 \cup H_3 \subseteq S \wedge_w \Delta$ , which leads to this lemma.

Let  $H$  be  $H_1 \cup H_2 \cup H_3$  and  $t = |H|$ . By Definition 1 and Definition 3,  $t$  is  $k+1$  or  $k+2$ , and in  $\pi_v^S$ ,  $s_1 \leq_v \dots \leq_v s_{t-3} <_v s_{t-2} =_v s_{t-1} =_v s_t <_v s_{t+1} \dots$ , and  $H = \{s_1, \dots, s_t\}$ .

Let  $k'$  be  $|S \wedge_s \Delta|$ . By Definition 7, for each site  $p \in S \wedge_s \Delta$ ,  $p <_v d(\Delta)$ . Therefore, there exists  $k'' \geq k'$  such that in  $\pi_v^S$ ,  $s_{k''-1} <_v s_{k''}$  and either  $s_{k''} = d(\Delta)$  or  $s_{k''} \leq_v d(\Delta)$ , implying that  $\{s_1, \dots, s_{k''-1}\} \subseteq S \wedge_w \Delta$ .

Since  $k'' > k$  and  $t = k+1$  or  $k+2$ , we have  $k'' > t$ ; otherwise,  $k'' = t$  or  $t-1$ , contradicting either  $s_{k''-1} <_v s_{k''}$  or  $s_{t-2} =_v s_{t-1} =_v s_t$ .

To conclude,  $H = \{s_1, \dots, s_t\} \subseteq \{s_1, \dots, s_{k''-1}\} \subseteq S \wedge_w \Delta$ . Thus  $v$  is a Voronoi vertex of  $V_k(S \wedge_w \Delta)$ .  $\square$

### 3.2 Computing the Voronoi Vertices of $V_k(S)$

Lemma 5 indicates that if for any  $\Delta \in \mathcal{V}_\beta^\Delta(R)$ ,  $|S \wedge_s \Delta| > k$ , then computing the Voronoi vertices of  $V_k(S)$  can be transformed into computing the Voronoi vertices of  $V_k(S \wedge_w \Delta)$  for each  $\Delta$ . Lemma 4 states that on average it takes two trials to generate a sample  $R$  such that  $|S \wedge_s \Delta| \geq |S|/(r-5)$ , where the size  $r$  of the random sample  $R$  is any sufficiently large constant. Therefore, if  $|S|/(r-5) > k$ , then we need two trials on average to generate a random sample that satisfies the conditions of Lemma 5. The condition  $|S \wedge_w \Delta| \leq \alpha|S|$  in Lemma 4 bounds the depth of the recursion. Following Clarkson [7], the algorithm to compute the Voronoi vertices of  $V_k(S)$  is summarized as follows:

- If  $|S|/(r-5) \leq k$ , compute the vertices of  $V_k(S)$  by the algorithm in Section 5.
- Otherwise ( $|S|/(r-5) > k$ )
  1. Choose  $R \subset S$  of size  $r$  until  $R$  satisfies the conditions of Lemma 4
    - (a) Construct  $V_\beta(R)$  by the algorithm in Section 4 and Compute  $\mathcal{V}_\beta^\Delta(R)$  from  $V_\beta(R)$  (Lemma 2).
    - (b) Check each trapezoid in  $\mathcal{V}_\beta^\Delta(R)$  to satisfy the conditions of Lemma 4.
  2. For each trapezoid  $\Delta \in \mathcal{V}_\beta^\Delta(R)$ 
    - (a) Recursively compute the Voronoi vertices of  $V_k(S \wedge_w \Delta)$ .
    - (b) Select vertices of  $V_k(S \wedge_w \Delta)$  that are vertices of  $V_k(S)$ .

### 3.3 Analysis

**Lemma 6.**  $V_k(S)$  can be computed from its Voronoi vertices in  $O(k(n-k) \log n)$  operations.

*Proof.* For points-sites, a vertex is uniquely defined by three sites [14]. Also for point-sites two vertices are adjacent iff their corresponding triples of sites have two sites in common. However, in the abstract setting, three sites may define one or two vertices and the adjacency property does not hold. Therefore, we cannot solve this problem by just using radix sort as it was done for point-sites [7].

Here, in the abstract setting, we use radix sort to extract for each bisector all Voronoi vertices that lie on it, in total  $O(|V|)$  operations, where  $V$  is the set of vertices in  $V_k(S)$ . We also assume the existence of a sufficiently large closed curve  $\Gamma$  such that no two bisectors intersect outside  $\Gamma$ .

Consider a set of  $m_J > 0$  Voronoi vertices that belong to bisector  $J$  (including the artificial Voronoi vertices formed by the intersection between  $V_k(S)$  and  $\Gamma$ ).  $m_J$  must be even; otherwise, at least one Voronoi vertex has no Voronoi edge. We can sort the  $m_J$  Voronoi vertices along one direction of  $J$  as  $v_1, v_2, \dots, v_{m_J}$  in  $O(m_J \log m_J)$  operations, and then link  $\overline{v_{2i-1}v_{2i}}$  for  $1 \leq i \leq m_J/2$  as Voronoi edges in  $O(m_J)$  operations. Therefore, we can compute all the Voronoi edges on  $J$  in  $O(m_J \log m_J)$  operations. Since  $|V|$  is  $O(k(n-k))$ , the total number of operations is  $O(|V|) + \sum_{J \in \mathcal{J}, m_J > 0} O(m_J \log m_J) = O(|V| \log |V|) = O(k(n-k) \log n)$ .  $\square$

**Theorem 1.**  $V_k(S)$  can be computed in expected  $O(kn^{1+\varepsilon})$  operations, where  $\varepsilon > 0$ , and the constant factor of the asymptotic bound depends on  $\varepsilon$ .

*Proof.* Recall that  $r$  is a sufficiently large constant,  $\alpha = O(\log r/r)$  and  $\beta = O(\log r/\log \log r)$ . There are two cases: (1) If  $|S|/(r-5) \leq k$ , then we use the algorithm from Section 5 to compute the vertices of the order- $k$  Voronoi diagram in expected  $O(n^2 2^{\alpha(n)} \log n)$  operations, i.e.  $O(r^2 k^2 \log^2 r \log^2 k)$ ; (2) If  $|S|/(r-5) > k$  then the algorithm proceeds as follows:

1. Choose a random sample that satisfies the conditions of Lemma 4. Do the check by constructing  $V_\beta(R)$  and computing  $\mathcal{V}_\beta^\Delta(R)$  from  $V_\beta(R)$ . The construction of  $V_\beta(R)$  takes expected  $O(r\beta^2 \log r)$  operations (see Section 4), and computing  $\mathcal{V}_\beta^\Delta(R)$  takes additional expected  $O(\beta(r-\beta) \log r)$  operations. The number of the trapezoids in  $\mathcal{V}_\beta^\Delta(R)$  is  $O(r\beta)$ , and the number of operations required to check the sample is  $O(nr\beta) \subset O(nr \log r)$ .
2. For each trapezoid in  $\mathcal{V}_\beta^\Delta(R)$  compute the order- $k$  vertices using recursion. The number of recursive calls is  $O(r\beta) \subset O(r \log r)$ . Each recursive call inputs  $O(\alpha n) = O(n \log r/r)$  sites and outputs  $O(\alpha nk)$  vertices. Therefore, the expected total number of operations required to validate each vertex of each recursive call is  $O(\alpha nkr \log r)$  which is  $O(nk \log^2 r)$ .

Therefore, the expected number  $t(n)$  of operations for computing the Voronoi vertices of  $V_k(S)$  is

$$t(n) \leq O(r^2 k^2 \log^2 r \log^2 k), \quad n \leq k(r-5)$$

$$t(n) \leq O(nr \log r) + O(nk \log^2 r) + O(r \log r)t(O(n \log r/r)), \quad n > k(r-5),$$

and the depth of the recursion is  $O(\log(n/k)/\log(r/\log r))$ .

Following [7, Lemma 6.4], if  $n$  tends to infinity,  $t(n)$  is  $O(kn^{1+\varepsilon})$ . Since  $V_k(S)$  can be constructed from the Voronoi vertices of  $V_k(S)$  in expected  $O(k(n-k) \log n)$  operations (Lemma 2),  $V_k(S)$  can be constructed in expected  $O(kn^{1+\varepsilon})$  operations.  $\square$

## 4 First Sub-Algorithm: Iterative Construction

The order- $k$  abstract Voronoi diagram can be computed iteratively similarly to point sites in the Euclidean metric [14]. The following lemma proves the main property used in the iterative construction.

**Lemma 7.** Let  $F$  be a face of  $VR_j(H, S)$  and let  $VR_j(H_i, S)$ ,  $1 \leq i \leq \ell$  be the adjacent regions. Then  $V_{j+1}(S) \cap F = V_1(Q) \cap F$ , where  $Q = \bigcup_{1 \leq i \leq \ell} H_i \setminus H$ .

*Proof.* We want to show  $V_1(Q) \cap F = V_{j+1}(S) \cap F$  which is equal to  $V_1(S \setminus H) \cap F$ .

Let  $x \in VR_1(s, S \setminus H) \cap F$ . For the sake of a contradiction assume  $s \notin Q$ . This means  $s <_x q$ , for any  $q \in Q$  and thus  $x \in VR_{j+1}(H \cup \{s\})$ . Let  $F'$  be the face of  $VR_{j+1}(H \cup \{s\})$  that contains  $x$ . Since  $s \notin Q$ ,  $F'$  does not intersect  $\partial F$ , implying that  $F' \cap V_j(S)$  is empty. This leads to a contradiction since  $F' \cap V_j(S) = F' \cap V_{n-1}(H \cup \{s\})$  and this is nonempty [3, Lemmata 12 and 13]. Hence  $V_1(S \setminus H) \cap F = V_1(Q) \cap F$  which finishes the proof.  $\square$

Lemma 7 implies that we can compute  $V_{j+1}(S)$  by partitioning each face of  $V_j(S)$  with the nearest-neighbor Voronoi diagram, which in turn can be computed using the algorithm in [12].

**Theorem 2.**  $V_k(S)$  can be computed in expected  $O(k^2 n \log n)$  operations.

## 5 Second Sub-Algorithm: Random Walk Method

We construct  $V_k(S)$  by computing  $\partial\text{VN}_k(p, S)$  for every  $p \in S$ , i.e., all the Voronoi edges of  $V_k(S)$  belonging to  $J(p, q)$ . Chazelle and Edelsbrunner [6] computed  $\partial\text{VN}_k(p, S)$  based on dynamic convex hulls and the fact that  $\text{VN}_k(p, S)$  is simply connected. However, dynamic convex hulls are not applicable in the abstract setting. Since  $\text{VN}_k(p, S)$  is simply connected, we can adopt Har-Peled's [9] random walk algorithm to compute  $\partial\text{VN}_k(p, S)$ .

$\partial\text{VN}_k(p, S)$  is a substructure of the arrangement of  $n-1$  bisectors  $\mathcal{J}(p) = \{J(p, q) \mid q \in S \setminus \{p\}\}$ , where the bisectors in  $\mathcal{J}(p)$  are not  $x$ -monotone, but they have constant number of vertical tangency points. Therefore, the structural complexities of the arrangement and its vertical decomposition are of the same asymptotic magnitude. We construct  $\partial\text{VN}_k(p, S)$  in the following way: (1) For each connected component of  $\partial\text{VN}_k(p, S)$  compute a starting point; (2) For each starting point, traverse the corresponding part of  $\partial\text{VN}_k(p, S)$ .

Lemma 8 states that starting points can be computed in  $O(n \log n)$  expected time. As we walk we can determine the next direction in  $O(1)$  time.

**Lemma 8.** *The starting points of  $\partial\text{VN}_k(p, S)$  for each of its connected components can be computed in total  $O(n \log n)$  expected time.*

Following [9], the expected number of operation required to compute the boundary of the  $k$ -neighborhood by the random walk is  $O(\lambda_{t+2}(n+m) \log n)$ , where  $t$  is the maximum number of intersections between two bisectors, and  $m$  is the complexity of  $\partial\text{VN}_k(p, S)$ . In the abstract case, we can show that  $t = 2$ , i.e. each pair of bisectors  $J(p, q)$  and  $J(p, r)$  in  $\mathcal{J}(p)$  intersect at most twice. Consider  $V_1(\{p, q, r\})$ . Axiom (A1) implies that each region in this diagram is connected, therefore  $V_1(\{p, q, r\})$  has at most two vertices. Thus,  $J(p, q)$  and  $J(p, r)$  intersect at most twice and  $t = 2$ .

The main difference between computing the zone in the original version of the algorithm [9] and computing  $\partial\text{VN}_k(p, S)$  is that the latter is additionally augmented by the vertical rays from the points of vertical tangency. However, since each bisector allows only a constant number of points of vertical tangency, the expected number of operations increases only by a constant factor.

**Theorem 3.**  $V_k(S)$  can be computed in expected  $O(n^2 2^{\alpha(n)} \log n)$  operations.

## References

1. Agarwal, P.K., de Berg, M., Matoušek, J., Schwarzkopf, O.: Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM Journal on Computing* **27**(3), 654–667 (1998)



2. Aurenhammer, F., Schwarzkopf, O.: A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. *International Journal of Computational Geometry and Applications* **2**(4), 363–381 (1992)
3. Bohler, C., Cheilaris, P., Klein, R., Liu, C.-H., Papadopoulou, E., Zavershynskiy, M.: On the complexity of higher order abstract Voronoi diagrams. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) *ICALP 2013, Part I*. LNCS, vol. 7965, pp. 208–219. Springer, Heidelberg (2013)
4. Boissonnat, J.D., Devillers, O., Teillaud, M.: A semidynamic construction of higher-order Voronoi diagrams and its randomized analysis. *Algorithmica* **9**, 329–356 (1993)
5. Chan, T.M.: Random sampling, halfspace range reporting, and construction of ( $\leq k$ )-levels in three dimensions. *SIAM Journal on Computing* **30**(2), 561–572 (1998)
6. Chazelle, B., Edelsbrunner, H.: An improved algorithm for constructing  $k$ th-order Voronoi Diagram. *IEEE Transactions on Computers* **36**(11), 1349–1454 (1987)
7. Clarkson, K.L.: New applications of random sampling in computational geometry. *Discrete and Computational Geometry* **2**(1), 195–222 (1987)
8. Gemsa, A., Lee, D.T., Liu, C.-H., Wagner, D.: Higher order city Voronoi diagrams. In: Fomin, F.V., Kaski, P. (eds.) *SWAT 2012*. LNCS, vol. 7357, pp. 59–70. Springer, Heidelberg (2012)
9. Har-Peled, S.: Taking a walk in a planar arrangement. *SIAM Journal on Computing* **30**(4), 1341–1367 (2000)
10. Klein, R.: *Concrete and Abstract Voronoi Diagrams*. LNCS, vol. 400. Springer, Heidelberg (1989)
11. Klein, R., Langetepe, E., Nilforoushan, Z.: Abstract Voronoi Diagrams Revisited. *Computational Geometry: Theory and Applications* **42**(9), 885–902 (2009)
12. Klein, R., Mehlhorn, K., Meiser, S.: Randomized Incremental Construction of Abstract Voronoi Diagrams. *Computational Geometry: Theory and Applications* **3**(1), 157–184 (1993)
13. Mehlhorn, K., Meiser, S., Ó'Dúnlaing, C.: On the Construction of Abstract Voronoi Diagrams. *Discrete and Computational Geometry* **6**(1), 211–224 (1991)
14. Lee, D.T.: On  $k$  Nearest Neighbor Voronoi Diagrams in the Plane. *IEEE Trans. Computers* **31**(6), 478–487 (1982)
15. Liu, C.-H., Lee, D.T.: Higher-order geodesic Voronoi diagrams in a polygonal domain with holes. In: *2013 ACM-SIAM Symposium on Discrete Algorithms*, pp. 1633–1645 (2013)
16. Liu, C.-H., Papadopoulou, E., Lee, D.T.: An output-sensitive approach for the  $L_1/L_\infty$   $k$ -Nearest-Neighbor Voronoi diagram. In: Demetrescu, C., Halldórsson, M.M. (eds.) *ESA 2011*. LNCS, vol. 6942, pp. 70–81. Springer, Heidelberg (2011)
17. Papadopoulou, E., Zavershynskiy, M.: On Higher Order Voronoi Diagrams of Line Segments. In: Chao, K.-M., Hsu, T.-s., Lee, D.-T. (eds.) *ISAAC 2012*. LNCS, vol. 7676, pp. 177–186. Springer, Heidelberg (2012)
18. Ramos, E.: On range reporting, ray shooting, and  $k$ -level construction. In: *15th ACM Symposium on Computational Geometry*, pp. 390–399 (1999)
19. Zavershynskiy, M., Papadopoulou, E.: A sweepline algorithm for higher order Voronoi diagrams. In: *Proc. 10th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*. IEEE-CS (2013)

# **Combinatorial Optimization I**

# Average-Case Complexity of the Min-Sum Matrix Product Problem

Ken Fong<sup>1</sup>(✉), Minming Li<sup>1</sup>, Hongyu Liang<sup>2</sup>, Linji Yang<sup>3</sup>,  
and Hao Yuan<sup>4</sup>

<sup>1</sup> Department of Computer Science,  
City University of Hong Kong, Hong Kong, China  
ken.fong@my.cityu.edu.hk, minming.li@cityu.edu.hk

<sup>2</sup> Facebook, Inc., Menlo Park, USA  
hongyuliang86@gmail.com

<sup>3</sup> Georgia Institute of Technology, Atlanta, USA  
ljyang@gatech.edu

<sup>4</sup> Bopu Technologies, Shenzhen, China  
hao@bopufund.com

**Abstract.** We study the average-case complexity of min-sum product of matrices, which is a fundamental operation that has many applications in computer science. We focus on optimizing the number of “algebraic” operations (i.e., operations involving real numbers) used in the computation, since such operations are usually expensive in various environments. We present an algorithm that can compute the min-sum product of two  $n \times n$  real matrices using only  $O(n^2)$  algebraic operations, given that the matrix elements are drawn independently and identically from some fixed probability distribution satisfying several constraints. This improves the previously best known upper-bound of  $O(n^2 \log n)$ . The class of probability distributions under which our algorithm works include many important and commonly used distributions, such as uniform distributions, exponential distributions, and folded normal distributions.

In order to evaluate the performance of the proposed algorithm, we performed experiments to compare the running time of the proposed algorithm with algorithms in [7]. The experimental results demonstrate that our algorithm achieves significant performance improvement over the previous algorithms.

## 1 Introduction

The min-sum product (also known as min-plus product, distance matrix product, and distance matrix multiplication) of two matrices is defined as follows: Given two  $n \times n$  real matrices  $A$  and  $B$ , the min-sum product of them, denoted by  $A \otimes B$ , is defined as a matrix  $C$  where

$$C_{i,j} \stackrel{\text{def}}{=} \min_k (A_{i,k} + B_{k,j}), \quad \text{for all } 1 \leq i, j \leq n. \quad (1)$$

---

This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 122512].

The min-sum product is a fundamental operation that has many applications in computer science. For example, it is well known that the min-sum product problem is closely related to the problem of computing all-pairs shortest paths in a graph [1], which is among the most fundamental and well-studied problems in the algorithm community. Another important application of the min-sum product is in performing MAP (maximum a posteriori) inference with graphical models [7, 13].

A naïve implementation for computing the min-sum product of two  $n \times n$  matrices requires  $O(n^3)$  time, which is too slow for a large  $n$ . Because of the importance of this problem, many algorithms were developed to improve the cubic time bound (see the work of Chan [4] and the references therein). Currently, the best-known worst-case algorithm is due to Han and Takaoka [11] with time complexity  $O(n^3 \log \log n / \log^2 n)$ , which is only slightly better than cubic time by a poly-logarithmic factor. Whether there exists a truly worst-case sub-cubic (i.e.,  $O(n^{3-\delta})$  for some positive constant  $\delta$ ) algorithm for the min-sum product problem is a long-standing open problem. The difficulty comes from the fact that the min-sum product is computed on a semiring structure, where there is no additive inverse defined over the min operator. Existing truly sub-cubic fast matrix multiplication algorithms (e.g., the Strassen algorithm [19] and the Coppersmith-Winograd algorithm [5]) only work on a ring structure. In fact, it is conjectured by many researchers that an  $n^{3-\Omega(1)}$  time algorithm does not exist for the min-sum product problem (see e.g. [4, 10, 11]).

In many practical applications of the min-sum product problem, the average-case time complexity becomes more interesting than worst-case complexity. As shown in a recent work of Felzenszwalb and McAuley [7], the min-sum product problem can be solved significantly faster than cubic time in the average case for some MAP inference applications in computer vision and natural language processing. (See also the work of McAuley and Caetano [13] for the related applications.) The algorithm of Felzenszwalb and McAuley [7] runs in expected  $O(n^2 \log n)$  time when the entries of the input matrices are independently drawn from a uniform distribution on  $[0, 1]$ . A more general algorithm of Takaoka [20] also runs in expected  $O(n^2 \log n)$  time under the endpoint independence model [2]. Although being almost tight, there is still an  $O(\log n)$  gap between the upper bound and the trivial lower bound of  $\Omega(n^2)$  (which is the time required to output the answer).

Since the computations involving real numbers are usually much costlier than that of integers, in real applications it is useful to consider the following restricted algebraic model of computation: Computations involving real numbers are restricted to adding two real numbers and comparing two real numbers. When talking about the *algebraic complexity*, we only count the costs for adding and comparing two real numbers; all other computations (like adding two integer variables, comparing two indices, etc.) are assumed to be cost-free.

For the min-sum product problem, this restricted computation model actually coincides with the decision tree model [8], which also counts only the number of comparisons and additions of the matrix elements (or edge weights in the

graph view) required in the computation. Thus, the result of [9] gives an  $O(n^{2.5})$  worst-case bound on the algebraic complexity of min-sum product, which is a substantial improvement on the  $n^{3-o(1)}$  complexity in the traditional model. However, this is not the case when considering the average case. The analyses of previous algorithms [7, 20] only give an  $O(n^2 \log n)$  bound on the average-case algebraic complexity, which is of the same order with the traditional average-case complexity. It is not clear from the previous studies whether this bound can be further improved.

*Our Contributions.* In this paper, we study the problem of computing the min-sum product of two random matrices under the aforementioned algebraic computation model. We show that the min-sum product of two  $n \times n$  matrices can be computed using only  $O(n^2)$  expected algebraic operations (i.e., adding or comparing two real numbers), when the elements of the two matrices are independently drawn from some fixed probability distribution satisfying several constraints (see Theorem 1). Thus, our result improves the algebraic complexity of computing the min-sum product of two matrices from  $O(n^2 \log n)$  [7, 20] to  $O(n^2)$ , which is clearly the best possible due to the trivial quadratic lower bound. The class of probability distributions under which our algorithm works include many important and commonly used distributions, e.g., the uniform distribution on  $[0, \vartheta]$  for any  $\vartheta > 0$ , exponential distributions, folded normal distributions. As mentioned before, the algebraic computation model actually coincides with the decision tree model. Therefore, as a by-product, the average-case decision tree complexity of min-sum product is shown to be  $O(n^2)$ , which matches the trivial  $\Omega(n^2)$  lower bound.

Besides the theoretical result we achieved, we re-implemented the algorithms in [7] and our algorithm in C++ to perform the comparison on the running time. For two matrices multiplication, the experimental results show that our algorithm achieves significant performance improvements over the previous algorithms, especially when  $n$  is large. Moreover, we also conducted the experiments with multiple matrices multiplication. Figure 2 shows that the improvement over algorithms in [7] is significant when  $m$  is small.

We note that a recent work of Peres et al. [17] solves the all-pairs shortest paths problem on a complete graph (or the  $G(n, p)$  model with moderately large  $p$ ) in expected  $O(n^2)$  time when the edge lengths are from the uniform distribution on  $[0, \vartheta]$ . Their work improves several previous results on the average-case complexity of the problem (e.g., [2, 12, 14–16, 18]). However, neither their algorithm nor the previous ones apply to our case.

## 2 Min-Sum Product of Two Matrices

Let  $A$  and  $B$  be two  $n$  by  $n$  matrices, and  $C = A \otimes B$  be their min-sum product. Assume that the entries of  $A$  and  $B$  are independent and identically distributed (i.i.d.) random variables drawn from some fixed probability distributions. Our goal is to efficiently compute  $C$ .

As introduced earlier, we focus on a restricted algebraic model of computation as follows: Computations that involve real numbers are restricted to

adding two real numbers and comparing two real numbers; no other computations are allowed for real numbers. The algebraic complexity of an algorithm is the (expected) number of algebraic operations, which only include the operation of adding two real numbers and that of comparing two real numbers; all other computations (like adding two integer variables, comparing two indices, etc) are cost-free. As noted in the introduction, this measure of complexity coincides with the well-studied decision tree complexity. Under this model, we have the following main theorem in this section.

**Theorem 1.** *The min-sum product of two  $n$  by  $n$  real matrices  $A$  and  $B$  can be computed using  $O(n^2)$  expected algebraic computations (more specifically,  $O(n^2)$  additions and comparisons of real numbers) if the elements of  $A$  and  $B$  are drawn independently from the same probability distribution, whose (cumulative) distribution function  $F$  satisfies all of the following three conditions:*

- $F$  is continuous;
- $\inf\{x|F(x) > 0\} = 0$ ;
- there exist two positive constants  $\lambda$  and  $\theta$ , such that  $F(x) \leq \lambda F(x/2)$  for  $0 \leq x \leq \theta$ .

Notice that the second condition in Theorem 1,  $\inf\{x|F(x) > 0\} = 0$ , means the distribution must be for non-negative random variables, and the left end-point of the support must be 0. Many popular probability distributions for non-negative random variables satisfy the conditions in Theorem 1, for example, the uniform distribution on  $[0, \vartheta]$  for any  $\vartheta > 0$ , exponential distributions, folded normal distributions, etc. Furthermore, we will later discuss how to support an even more general class of distributions.

We present our algorithm as Algorithm 1, which can be considered as a refined version of the algorithm of Felzenszwalb and McAuley [7]. The major differences are the introduction of the matrix  $D$  in Algorithm 1 and the selection of the  $6n^2/\log n$  smallest matrix entries in  $A$  and  $B$ . Notice that in the algorithm,  $\hat{C}$  represents the min-sum product computed by the algorithm, and  $C$  (without the hat) represents the correct min-sum product of  $A$  and  $B$ .

Here we sketch the basic ideas. The elements of matrices are independently drawn from commonly used distributions, includes normal distribution, exponential distribution, etc. The algorithm maintains an extra matrix  $D$  to keep track of whether the elements in  $\hat{C}$  are minimized. Similar to algorithm of Felzenszwalb and McAuley [7], all elements in  $\hat{C}$  are initialized to infinity. The algorithm uses linear-time selection algorithm to select the  $r$  smallest matrix entries in  $A$  and  $B$ . Each time, it finds the smallest element from  $r$  entries and check whether it is greater than or equal to the source-sink pair in  $\hat{C}$ . If this is the case, the algorithm will mark this pair in  $D$  as TRUE, as it cannot be minimized in the remaining iterations. Otherwise, the relaxation is performed on the lengths of source-sink pair. After processing all  $r$  elements, if there exists any element not marked as true in  $D$ , it will use the naïve algorithm to find out the correct values for those elements. The algorithm terminates when all elements in matrix  $D$  are marked as TRUE.

**Algorithm 1.** Computing the Min-Sum Product of Two Matrices**Input:**  $A$  and  $B$ , which are both  $n$  by  $n$  matrices.**Output:**  $\hat{C}$ , which should be equal to  $C = A \otimes B$  when the algorithm exits.

```

1 Use a linear-time selection algorithm (e.g. [3]) to select  $r = \frac{6n^2}{\log n}$  smallest
  numbers from  $A$  and  $B$  in  $O(n^2)$  time. Ties are broken arbitrarily.
2 Sort these  $r$  numbers in  $O(r \log r) = O(n^2)$  time, breaking ties arbitrarily, to get
  a non-decreasing sequence of numbers  $S_1, S_2, \dots, S_r$ , where  $S_p$  denotes the  $p$ -th
  smallest number in  $A$  and  $B$ . (Note that we can now access  $S_p$  for  $1 \leq p \leq r$ .)
3 Initialize  $\hat{C}_{i,j} \leftarrow \infty$  for  $1 \leq i, j \leq n$ .
4 Initialize an  $n$  by  $n$  Boolean matrix  $D$  by setting  $D_{i,j} \leftarrow \text{FALSE}$  for  $1 \leq i, j \leq n$ .
  Here  $D_{i,j} = \text{TRUE}$  means  $\hat{C}_{i,j} = C_{i,j}$ , i.e., the value of  $C_{i,j}$  is correctly
  computed. Note that once  $D_{i,j}$  is set to TRUE, it will not change back to FALSE
  in the later execution of the algorithm.
5 Initialize  $L[k] \leftarrow \emptyset$  and  $R[k] \leftarrow \emptyset$  for all  $1 \leq k \leq n$ .
6 for  $p = 1$  to  $r$  do
7   if  $S_p$  is  $A_{i,k}$  for some  $i$  and  $k$  then
8     |  $L[k] \leftarrow L[k] \cup \{S_p\}$ .
9   else if  $S_p$  is  $B_{k,j}$  for some  $k$  and  $j$  then
10    |  $R[k] \leftarrow R[k] \cup \{S_p\}$ .
11    //In the following, we assume  $S_p$  is  $A_{i,k}$ . The other case where  $S_p$  is  $B_{k,j}$  is
    totally analogous: just replace the next line with “for each  $A_{i,k} \in L[k]$ ”.
12    for each  $B_{k,j} \in R[k]$  do
13      | if  $D_{i,j} = \text{FALSE}$  then
14        | | if  $S_p \geq \hat{C}_{i,j}$  then
15          | | |  $D_{i,j} \leftarrow \text{TRUE}$ ;
16          | | | If all entries of  $D$  are TRUE, then exit the algorithm and return
17          | | |  $\hat{C}$ .
18          | | else
19          | | |  $\hat{C}_{i,j} \leftarrow \min\{\hat{C}_{i,j}, A_{i,k} + B_{k,j}\}$ 
19 if there exists a FALSE in  $D$  then
20 | Use a naïve  $O(n^3)$  algorithm to compute  $\hat{C}$ .
21 return  $\hat{C}$ 

```

**Lemma 1.** During the execution of Algorithm 1, when  $D_{i,j}$  is TRUE, we have  $\hat{C}_{i,j} = C_{i,j}$ .

*Proof.* This is because of the monotonic nondecreasing property of  $S$ : once  $D_{i,j}$  is changed from FALSE to TRUE,  $S_{p'} + S_{q'} \geq S_{p'} \geq S_p \geq \hat{C}_{i,j} \geq C_{i,j}$  for any  $p' \geq p$  and any  $q'$ . So  $\hat{C}_{i,j}$  will not be relaxed by any  $S_{p'} + S_{q'}$  in the later execution of the algorithm, which implies that  $\hat{C}_{i,j} = C_{i,j}$ .  $\square$

Because of Lemma 1, if all the entries of  $D$  are TRUE at the end of the algorithm, then  $\hat{C}$  is computed correctly. Otherwise, lines 19–20 of the algorithm will guarantee the correctness of the computed  $\hat{C}$ .

Now we analyze the algebraic complexity of Algorithm 1. Clearly lines 1–5 take  $O(n^2)$  time, and hence require only  $O(n^2)$  algebraic computations. We need the following Proposition 1 and Proposition 2. If these two propositions hold, the expected number of comparisons and additions is  $O(n^2 + n^3 \cdot 1/n) = O(n^2)$ , which proves Theorem 1.

**Proposition 1.** *For a fixed probability distribution of Theorem 1, lines 6–18 of Algorithm 1 require  $O(n^2)$  expected number of additions and comparisons of real numbers.*

The algebraic computations only occur in lines 14 and 18. Let  $Q_1$  denote the number of algebraic computations in line 14, and  $Q_2$  denote that in line 18. When executing line 14, if the condition  $S_p \geq \hat{C}_{i,j}$  holds, then one of the entries of  $D$  will change from FALSE to TRUE; if the condition does not hold, line 18 will be executed. Since there are only  $n^2$  entries of  $D$ , we have  $Q_1 \leq O(n^2) + Q_2$ , and hence the total algebraic computations is bounded by  $2Q_2 + O(n^2)$ . The following Lemma 2 characterizes the behavior of the algebraic computations in line 18.

**Lemma 2.** *Right before executing line 18 of Algorithm 1, we have  $A_{i,k} \leq C_{i,j}$  and  $B_{k,j} \leq C_{i,j}$ .*

*Proof.* We will prove it by contradiction. Assume that the lemma does not hold, then we have either  $A_{i,k} > C_{i,j}$  or  $B_{k,j} > C_{i,j}$ , which implies  $S_p > C_{i,j}$ . In this case, we must have  $\hat{C}_{i,j} = C_{i,j}$ , because the  $A_{i,k'}$  and  $B_{k',j}$  that achieve  $C_{i,j} = A_{i,k'} + B_{k',j}$  must have been tried before the execution of the line due to the monotonicity of  $S_p$ . However,  $\hat{C}_{i,j} = C_{i,j}$  and  $S_p > C_{i,j}$  imply that  $S_p > \hat{C}_{i,j}$ , which is impossible, because line 14 will not allow the algorithm to branch into line 18 under such a condition. This contradiction implies the correctness of the lemma.  $\square$

Based on Lemma 2, the expected number of algebraic computations in line 18 is bounded by the expected size of the set  $\{(i, k, j) \mid A_{i,k} \leq C_{i,j} \wedge B_{k,j} \leq C_{i,j}\}$ . Let  $A$  denote the size of this set. For  $1 \leq i, k, j \leq n$ , let  $X_{i,k,j}$  be a random variable, where  $X_{i,k,j} = 1$  if  $A_{i,k} \leq C_{i,j} \wedge B_{k,j} \leq C_{i,j}$ , and  $X_{i,k,j} = 0$  otherwise. We have  $A = \sum_{i,k,j} X_{i,k,j}$ , and

$$\mathbf{E}[A] = \mathbf{E}\left[\sum_{i,k,j} X_{i,k,j}\right] = \sum_{i,k,j} \mathbf{E}[X_{i,k,j}] = \sum_{i,k,j} \Pr(A_{i,k} \leq C_{i,j} \wedge B_{k,j} \leq C_{i,j}).$$

Then, the following Lemma 3 suffices to establish Proposition 1.



**Lemma 3.** *For a fixed probability distribution with distribution function  $F$  as in Theorem 1, for any  $1 \leq i, j, k \leq n$ , we have*

$$\Pr(A_{i,k} \leq C_{i,j} \wedge B_{k,j} \leq C_{i,j}) \leq \lambda^2 \left(1 + \frac{1}{F^2(\theta)}\right) \cdot \frac{1}{n},$$

where the two positive constants  $\lambda$  and  $\theta$  are specified as in Theorem 1 for  $F$ .

As  $\lambda$  and  $\theta$  are both constants independent of  $n$ , by

$$\mathbf{E}[A] = \sum_{i,j,k} \Pr(A_{i,k} \leq C_{i,j} \wedge B_{k,j} \leq C_{i,j}) \leq n^3 \cdot O(1/n) = O(n^2).$$

Thus the total number of algebraic computations in Algorithm 1 is at most  $2\mathbf{E}[A] + O(n^2) = O(n^2)$ , which proves Proposition 1. Due to space limit, the proof of this lemma is omitted.

**Proposition 2.** *For a fixed probability distribution of Theorem 1, the probability to execute line 20 of Algorithm 1 is  $O(1/n)$ .*

*Proof.* Fix  $1 \leq i, j \leq n$ . We first observe that  $D_{i,j}$  is TRUE if there exists  $t_2 > t_1 \geq 0$  such that all of the following three conditions hold:

1.  $C_{i,j} \leq t_1$ .
2. There exists  $1 \leq k \leq n$  such that  $t_1 < A_{i,k} \leq t_2$  or  $t_1 < B_{k,j} \leq t_2$ .
3. There exists at least  $2n^2 - r = 2n^2 - 6n^2 / \log n$  edges with length larger than  $t_2$ .

In fact, conditions 2 and 3 ensure that  $A_{i,k}$  or  $B_{k,j}$  is among the  $r$  chosen edges. Consider the iteration in which  $S_p$  is this edge. Conditions 1 and 2 together guarantee that  $\hat{C}_{i,j}$  is updated to  $t_1$  before this iteration, and at this iteration  $D_{i,j}$  is set to TRUE by lines 14 and 15.

Next we show that the probability that such  $t_1$  and  $t_2$  do not exist is at most  $O(1/n^3)$ . By a simple union bound over the  $n^2$  possible pairs of  $(i, j)$ , this implies Proposition 2.

Because  $F$  is continuous, there exist  $r_2 > r_1 \geq 0$  such that  $F(r_1) = \frac{\log^2 n}{\sqrt{n}}$  and  $F(r_2) = \frac{2\log^2 n}{\sqrt{n}}$ . We consider two cases.

**Case 1:**  $r_1 \leq \theta$  (recall that  $\theta$  is the positive constant introduced in the statement of Theorem 1). We let  $t_1 = r_1$  and  $t_2 = r_2$ . The length of every edge falls in the range  $(t_1, t_2)$  with probability  $F(t_2) - F(t_1) = \frac{\log^2 n}{\sqrt{n}}$ . Thus condition 2 fails with probability at most

$$\left(1 - \frac{\log^2 n}{\sqrt{n}}\right)^{2n} \leq e^{-2\sqrt{n}\log^2 n} \leq O(n^{-3}).$$

If condition 3 fails, then there exists at least  $r$  edges with length at most  $t_2$ . Therefore, the probability that condition 3 fails is at most

$$\begin{aligned} \binom{2n^2}{r} (F(t_2))^r &= \binom{2n^2}{6n^2/\log n} \left( \frac{2 \log^2 n}{\sqrt{n}} \right)^{6n^2/\log n} \\ &\leq 2^{2n^2} \cdot \left( \frac{n^{1/10}}{\sqrt{n}} \right)^{6n^2/\log n} \\ &= 2^{2n^2} \cdot \left( 2^{-\frac{2}{5} \log n} \right)^{6n^2/\log n} \\ &= 2^{2n^2} \cdot 2^{-\frac{12}{5} n^2} \\ &\leq O(n^{-3}), \end{aligned}$$

Now consider condition 1. If condition 1 fails, then for all  $1 \leq k \leq n$ ,  $A_{i,k} + B_{k,j} > t_1$ . We have

$$\Pr(A_{i,k} + B_{k,j} \leq t_1) \geq \Pr(A_{i,k} \leq t_1/2 \wedge B_{k,j} \leq t_1/2) = F(t_1/2)^2 \geq (F(t_1)/\lambda)^2 = \frac{\log^4 n}{\lambda^2 n},$$

where we use  $F(x) \leq \lambda F(x/2)$  when  $x \leq \theta$ , and that  $t_1 \leq \theta$ . Then,

$$\Pr(A_{i,k} + B_{k,j} > t_1) \leq 1 - \frac{\log^4 n}{\lambda^2 n}.$$

Hence, we have

$$\Pr(C_{i,j} > t_1) \leq \left( 1 - \frac{\log^4 n}{\lambda^2 n} \right)^n \leq e^{-\Omega(\log^4 n)} \leq O(n^{-3}).$$

By the union bound, the probability that at least one condition fails is at most  $O(1/n^3)$ .

**Case 2:**  $r_1 > \theta$ . In this case we let  $t_1 = \theta$  and  $t_2 = r_2$ . Similar to Case 1, condition 3 fails with probability at most  $O(n^{-3})$ . Since  $F(t_2) - F(t_1) = F(r_2) - F(\theta) \geq F(r_2) - F(r_1) = \frac{\log^2 n}{\sqrt{n}}$ , we still have that condition 2 fails with probability at most  $(1 - \frac{\log^2 n}{\sqrt{n}})^{2n} \leq O(n^{-3})$ .

Now we consider condition 1. Similar as before, we have  $(\forall 1 \leq k \leq n) A_{i,k} + B_{k,j} > t_1$  if condition 1 fails. We also have

$$\Pr(A_{i,k} + B_{k,j} \leq t_1) \geq \Pr(A_{i,k} \leq \theta/2 \wedge B_{k,j} \leq \theta/2) = (F(\theta/2))^2.$$

Thus,  $\Pr(A_{i,k} + B_{k,j} > t_1) \leq 1 - (F(\theta/2))^2$ , and

$$\Pr(C_{i,j} > t_1) \leq \Pr(\forall 1 \leq k \leq n, A_{i,k} + B_{k,j} > t_1) \leq (1 - (F(\theta/2))^2)^n \leq e^{-(F(\theta/2))^2 n}.$$

As  $\theta$  is a positive constant, using the second condition of Theorem 1, we have  $F(\theta/2) = \Omega(1)$ , and thus  $\Pr(C_{i,j} > t_1) \leq e^{-\Omega(n)} \leq O(n^{-3})$ . Again, using the union bound, we know the probability that at least one condition fails is at most  $O(n^{-3})$ . This completes the proof of Proposition 2.  $\square$

### 3 Experiments

In this section, we use experiments to validate the effectiveness of our algorithm. Since the algorithms in [7] is considered as the previous best algorithm to compute MSP, we only compare the running time performance of our algorithm with the algorithms in [7]. There are two algorithms in [7] which are the algorithm with integer queue and the algorithm without integer queue. All three algorithms are implemented in C++ using the visual C++ compiler on an Intel i7 machine running Windows 7 operating system.

The elements of all the input matrices are taken from four kinds of random distributions which are uniform distribution of  $[0,1]$ , exponential distribution of  $[0,1]$ , normal distribution of  $[0,10]$ , and gamma distribution of  $[0,10]$ . For each algorithm, we ran 10 test cases for  $n = 100, 200, \dots$  up to  $n=1000$ . For each test case, we tested 100 input instances.

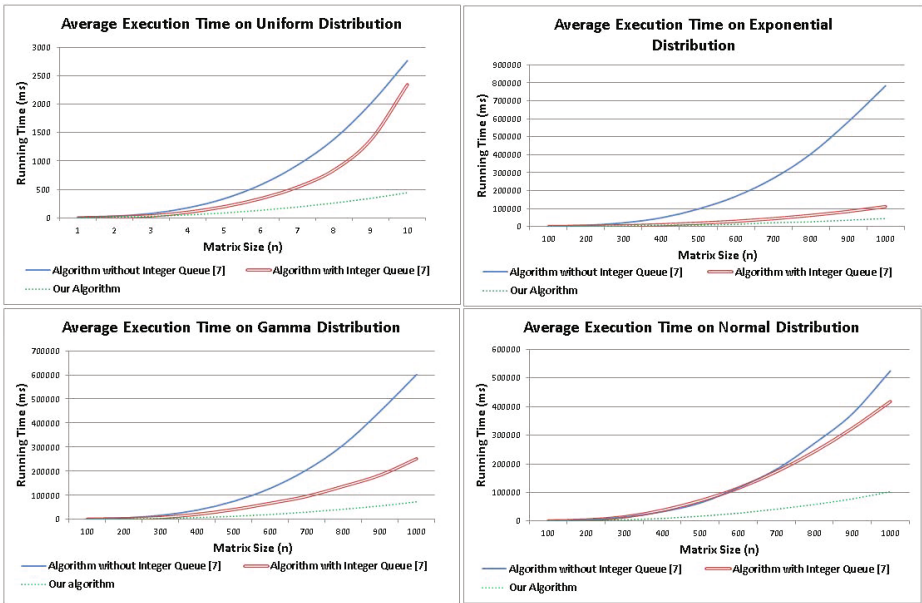


Fig. 1. Average Execution Time of Min-Sum Algorithms

Figure 1 shows the average execution time among these 100 instances for each test case. The result validates that our algorithm has better performance on random inputs. Specifically, when  $n$  is larger, our algorithm achieved better performance.

We also tested the performance of min-sum product on multiple matrices using the three algorithms as the basis. Besides matrix size  $n$ , a new parameter

$m$ , the number of matrices is introduced as one further dimension in the experiments. Hence, for each algorithm, with the same setting above, we added one further dimension  $m$  where  $m = 3, 4, \dots, 10$ .

Without modifying the algorithm, we implemented min-sum product of multiple matrices in a binary-tree-like association order. For example, assume the number of matrices  $m = 5$ , the matrices are denoted as  $A, B, C, D$ , and  $E$  respectively. The product order will be  $(A \otimes B) \otimes ((C \otimes D) \otimes E)$ . This way, more operations can be done on truly random input matrices.

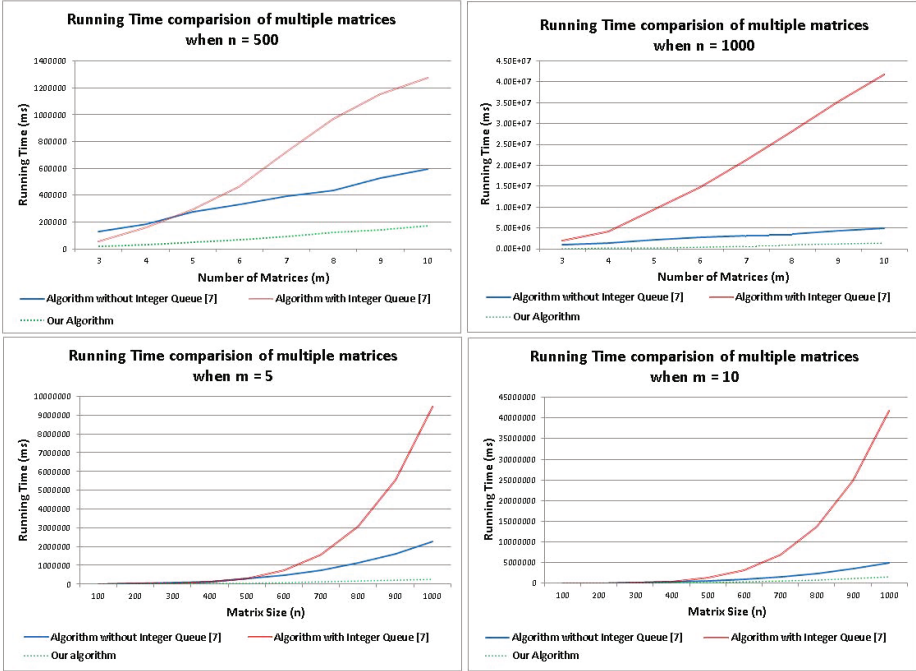


Fig. 2. Average Running Time of Min-Sum Algorithms with Multiple Matrices

We fix one dimension to see how the performance changes with the other dimension. The four diagrams in Figure 2 show the performance changes with  $m=5, 10, n=500, 1000$  respectively. The results show that our algorithm also performs better than algorithms in [7]. However, for algorithms in [7], they have different behaviors. When  $n = 500, m < 5$ , the algorithm with integer queue performs better than the algorithm without integer queue, same as the experiments for two matrices. However, when  $m \geq 5$ , the behavior is reversed. The algorithm without integer queue achieves better result as  $m$  increases.

In order to investigate the main reason of the poor performance of the algorithm with integer queue when  $m$  increases, we need to look at how integer queue is implemented. For the integer queue data structure, in the beginning,

the maximum value in matrix  $A$  and  $B$  will be selected as the parameter  $k$ . Then elements in matrix  $C$  are initialized to  $2k$ . For our experiment, all matrix elements are in the range of  $[0,1]$ , and hence  $k$  will be 1, and  $2k$  will be 2. Suppose we have a matrix with size  $n = 10$ . We need to create the buckets as follows:

bucket 0 stores values in  $[0, 0.01)$ ,  
 ...  
 bucket 199 stores values in  $[1.99, 2)$ ,  
 bucket 200 stores anything  $\geq 2$ .

Totally there will be  $2n^2$  buckets. For min-sum product on two matrices, since both matrices are truly-random matrices, each bucket will not store many items on average. However, for multiple matrices case, as the input is not truly random, the matrix elements will have a low chance to be evenly distributed to the buckets. Thus, some of the buckets may contain lots of elements. Since the algorithm will extract the minimum value from the bucket before computation, if there are lots of items in a single bucket, the performance may degrade. This is the main reason that the algorithm using integer queue performs worse when  $m$  is larger.

## 4 Conclusion

In this paper, we have investigated the average-case complexity of computing the min-sum product of matrices, and improve previously known results under the algebraic complexity setting. It remains an interesting question whether  $O(n^2)$  expected algebraic computations suffice also for the more general case, where a constant number of (larger than two) matrices are given as inputs.

Moreover, the bottleneck of our algorithm is due to the selection and sorting of the  $r$  smallest elements. If  $r$  can be further reduced, the running time of the bottleneck can be improved. But this may cause more elements in  $D$  to remain FALSE due to elements not minimized, and require to use the naïve algorithm to retrieve the correct values. Hence, if we choose the small  $r$ , then it may take longer time to execute the naïve algorithm. So it remains the question what is the best value of  $r$  to choose in order to improve the running time of the algorithm. Therefore we can consider this as our future works.

Besides the theoretical results we achieved, we run experiments to perform the running time comparison of our algorithm and algorithms in [7]. The experiments show that our algorithm achieves better result on random inputs. In addition, we also run experiments with multiple matrices multiplication. The results show that the improvement over algorithms in [7] is significant when  $m$  is small.

## References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)
2. Bloniarz, P.A.: A shortest-path algorithm with expected time  $O(n^2 \log n \log^* n)$ . SIAM Journal on Computing **12**(3), 588–600 (1983). <http://link.aip.org/link/?SMJ/12/588/1>

3. Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. *Journal of Computer and System Sciences* **7**(4), 448–461 (1973). <http://www.sciencedirect.com/science/article/pii/S0022000073800339>
4. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing* **39**(5), 2075–2089 (2010)
5. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation* **9**(3), 251–280 (1990). <http://www.sciencedirect.com/science/article/pii/S0747717108800132>
6. David, H.A., Nagaraja, H.N.: *Order Statistics*. Wiley, Hoboken (2003)
7. Felzenszwalb, P.F., McAuley, J.J.: Fast inference with min-sum matrix product. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **33**(12), 2549–2554 (2011)
8. Fredman, M.L.: On the decision tree complexity of the shortest path problems. In: *Proc. 16th FOCS*, pp. 98–99 (1975)
9. Fredman, M.L.: New bounds on the complexity of the shortest path problems. *SIAM Journal on Computing* **5**(1), 83–89 (1976)
10. Han, Y.: An  $O(n^3(\log \log n / \log n)^{5/4})$  time algorithm for all pairs shortest paths. *Algorithmica* **51**(4), 428–434 (2008)
11. Han, Y., Takaoka, T.: An  $O(n^3 \log \log n / \log^2 n)$  time algorithm for all pairs shortest paths. In: *Proc. 13th SWAT* (2012)
12. Karger, D., Koller, D., Phillips, S.: Finding the hidden path: time bounds for all-pairs shortest paths. *SIAM Journal on Computing* **22**(6), 1199–1217 (1993)
13. McAuley, J.J., Caetano, T.S.: Exploiting within-clique factorizations in junction-tree algorithms. *Journal of Machine Learning Research - Proceedings Track* **9**, 525–532 (2010)
14. McGeoch, C.: All-pairs shortest paths and the essential subgraph. *Algorithmica* **13**(5), 426–441 (1995)
15. Mehlhorn, K., Priebe, V.: On the all-pairs shortest-path algorithm of Moffat and Takaoka. *Random Structures and Algorithms* **10**(1–2), 205–220 (1997)
16. Moffat, A., Takaoka, T.: An all pairs shortest path algorithm with expected time  $O(n^2 \log n)$ . *SIAM Journal on Computing* **16**(6), 1023–1031 (1987)
17. Peres, Y., Sotnikov, D., Sudakov, B., Zwick, U.: All-pairs shortest paths in  $O(n^2)$  time with high probability. In: *Proc. 51th FOCS*, pp. 663–672 (2010)
18. Spira, P.M.: A new algorithm for finding all shortest paths in a graph of positive arcs in average time  $O(n^2 \log^2 n)$ . *SIAM Journal on Computing* **2**(1), 28–32 (1973). <http://link.aip.org/link/?SMJ/2/28/1>
19. Strassen, V.: Gaussian elimination is not optimal. *Numerische Mathematik* **13**(4), 354–356 (1969). <http://dx.doi.org/10.1007/BF02165411>
20. Takaoka, T.: Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electronic Notes in Theoretical Computer Science* **61**, 191–200 (2002). <http://www.sciencedirect.com/science/article/pii/S1571066104003135>

# Efficiently Correcting Matrix Products

Leszek Gąsieniec<sup>1</sup>, Christos Levcopoulos<sup>2</sup>(✉), and Andrzej Lingas<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Liverpool,  
Peach Street, Liverpool L69 7ZF, UK  
L.A.Gasieniec@liverpool.ac.uk

<sup>2</sup> Department of Computer Science, Lund University, 22100 Lund, Sweden  
{Christos.Levcopoulos, Andrzej.Lingas}@cs.lth.se

**Abstract.** We study the problem of efficiently correcting an erroneous product of two  $n \times n$  matrices over a ring. We provide a randomized algorithm for correcting a matrix product with  $k$  erroneous entries running in  $\tilde{O}(\sqrt{kn}^2)$  time and a deterministic  $\tilde{O}(kn^2)$ -time algorithm for this problem (where the notation  $\tilde{O}$  suppresses polylogarithmic terms in  $n$  and  $k$ ).

**Keywords:** Matrix multiplication · Matrix product verification · Correction algorithms · Randomized algorithms

## 1 Introduction

Matrix multiplication is a basic operation used in many sciences and engineering. There are several potential reasons for erroneous computational results, in particular erroneous matrix products. They include software bugs, computational errors by logic circuits and bit-flips in memory. If the computation is done by remote computers or by parallel processors, then some errors in the computed result might also be introduced due to faulty communication.

In 1977, Freivalds presented a randomized algorithm for verifying if a matrix  $C'$  is the matrix product of two  $n \times n$  matrices  $A$  and  $B$ , running in  $O(n^2)$  time [7]. His algorithm has been up today one of the most popular examples showing the power of randomization.

In spite of extensive efforts of the algorithmic community to derandomize it without substantially increasing its time complexity, one has solely succeeded partially, either decreasing the number of random bits to a logarithmic one [2, 9, 12] or using exponentially large numbers and the unrealistic BSS computational model [10]. One can argue that the latter solutions in different ways hide additional  $O(n)$  factors. By the way, if one can use quantum devices then even an  $O(n^{5/3})$ -time verification of  $n \times n$  matrix product over an integral domain is possible [1].

Interestingly, the problem of verifying matrix products over the  $(\min, +)$  semi-ring seems to be much harder than that over an arbitrary ring. Namely, it

---

Christos Levcopoulos and Andrzej Lingas: Research supported in part by Swedish Research Council grant 621-2011-6179.

admits a truly subcubic algorithm if and only if there is a truly subcubic algorithm for the all-pairs shortest path problem on weighted digraphs (APSP) [15].

Freivalds' algorithm has also pioneered a new subarea of the so called certifying algorithms [11]. Their purpose is to provide besides the output a certificate or easy to verify proof that the output is correct. The computational cost of the verification should be substantially lower than that incurred by recomputing the output (perhaps using a different method) from scratch.

In 1977, when Freivalds published his algorithm, the asymptotically fastest known algorithm for arithmetic matrix multiplication was that due to Strassen running in  $O(n^{2.81})$  time [13]. Since then the asymptotic running time of fast matrix multiplication algorithms has been gradually improved to  $O(n^{2.3728639})$  at present [3, 8, 14] which is still substantially super-quadratic.

In this paper, we go one step further and consider a more complex problem of not only verifying a computational result but also correcting it if necessary. Similarly as Freivalds, as a subject of our study we choose matrix multiplication.

Our approach is very different from that in fault tolerant setting, where one enriches input in order to control the correctness of computation (e.g., by check sums in the so called ABFT method) [5, 16, 17]. Instead, we use here an approach resembling methods from Combinatorial Group Testing where one keeps testing larger groups of items in search for multiple targets, see, e.g. [4, 6].

First, we provide a simple deterministic algorithm for correcting an  $n \times n$  matrix product  $C'$  over a ring, with at most one erroneous entry, in  $O(n^2)$  time. It can be regarded as a deterministic version of Freivalds' algorithm (Section 3). Next, we extend the aforementioned algorithm to include the case when  $C'$  contains at most  $k$  erroneous entries. The extension relies on distributing erroneous entries of  $C'$  into distinct submatrices by deterministically moving the columns of  $C'$  and correspondingly the columns of  $B$ . The resulting deterministic algorithm runs in  $\tilde{O}(k^2 n^2)$  time, where the notation  $\tilde{O}$  suppresses polylogarithmic terms in  $n$  and  $k$  (Section 4). Then we show how to reduce the time bound to  $\tilde{O}(kn^2)$  by applying this shuffling approach first with respect to the columns and then with respect to the rows of  $C'$ . In the same section, we discuss also a slightly randomized version of the aforementioned algorithm running in  $\tilde{O}(\sqrt{kn^2})$  expected time using  $O(\log^2 k + \log k \log \log n)$  random bits. For small  $k$ , this is less than the logarithmic in  $n$  number of random bits used in the best known  $O(n^2)$ -time verification algorithms for matrix multiplication obtained by a partial derandomization of Freivalds' algorithm [2, 9, 12]. Finally, in Section 5, we present a faster randomized algorithm for correcting  $C'$  in  $O(\sqrt{kn^2} \log n)$  time almost surely (i.e., with probability at least  $1 - n^{-\alpha}$  for any constant  $\alpha \geq 1$ ), where  $k$  is the non-necessarily known number of erroneous entries of  $C'$ . A slight modification of this algorithm runs in  $O(\sqrt{kn^2})$  expected time provided that the number of erroneous entries is known. Features of our algorithms are summarized in Table 1. Note that none of them subsumes any other one in all aspects. We conclude with Final Remarks, where we discuss how the  $O(\sqrt{kn^2})$ -expected-time algorithm from Section 5 and the slightly randomized algorithm from Section 4 can also be adjusted to the situation when the number of erroneous entries is unknown.



**Table 1.** The characteristics and time performances of the algorithms for correcting an  $n \times n$  matrix product with at most  $k$  erroneous entries presented in this paper. The issue of adapting the algorithms presented in the second and fourth row (not counting the title row) to unknown  $k$  is discussed in Final Remarks.

# errors = $e \leq k$	deterministic/randomized	time complexity
$k$ known	deterministic	$\tilde{O}(kn^2)$ time
$k = e$ , known	$O(\log^2 k + \log k \log \log n)$ random bits	$\tilde{O}(\sqrt{kn^2})$ expected time
$k = e$ , unknown	randomized	$O(\sqrt{kn^2} \log n)$ almost surely
$k = e$ , known	randomized	$O(\sqrt{kn^2})$ expected time

## 2 Preliminaries

Let  $(U, +, \times)$  be a semi-ring. For two  $n$ -dimensional vectors  $a = (a_0, \dots, a_{n-1})$  and  $b = (b_0, \dots, b_{n-1})$  with coordinates in  $U$  their dot product  $\sum_{i=0}^{n-1} a_i \times b_i$  over the semi-ring is denoted by  $a \odot b$ .

For an  $p \times q$  matrix  $A = (a_{ij})$  with entries in  $U$ , its  $i$ -th row  $(a_{i1}, \dots, a_{in})$  is denoted by  $A(i, *)$ . Similarly, the  $j$ -th column  $(a_{1j}, \dots, a_{nj})$  of  $A$  is denoted by  $A(*, j)$ . Given another  $q \times r$  matrix  $B$  with entries in  $U$ , the matrix product  $A \times B$  of  $A$  with  $B$  over the semi-ring is a matrix  $C = (c_{ij})$ , where  $c_{ij} = A(i, *) \odot B(*, j)$  for  $1 \leq i, j \leq n$ .

## 3 Correcting a Matrix Product with a Single Error

Given two matrices  $A$ ,  $B$  of size  $p \times q$  and  $q \times r$ , respectively, and their possibly erroneous  $p \times r$  matrix product  $C'$  over a ring, Freivalds' algorithm picks uniformly at random a vector in  $\{0, 1\}^r$  and checks if  $A(Bx^T) = C'x^T$ , where  $x^T$  stands for a transpose of  $x$ , i.e., the column vector corresponding to  $x$  [7]. For  $i = 1, \dots, p$ , if the  $i$ -th row of  $C'$  contains an erroneous entry, the  $i$ -th coordinates of the vectors  $A(Bx^T)$  and  $C'x^T$  will differ with probability at least  $1/2$ .

In the special case, when  $C'$  contains a single error, we can simply deterministically set  $x$  to the vector  $(1, \dots, 1) \in \{0, 1\}^r$  in the aforementioned Freivalds' test. The vectors  $A(Bx^T)$ ,  $C'x^T$  will differ in exactly one coordinate whose number equals the number of the row of  $C'$  containing the single erroneous entry. (Note that the assumption that there is only one error is crucial here since otherwise two or more errors in a row of  $C'$  potentially could cancel out their effect so that the dot product of the row with  $x$ , which in this case is just the sum of entries in the row, would be correct.) Then, we can simply compute the  $i$ -th row of the matrix product of  $A$  and  $B$  in order to correct  $C'$ .

The time complexity is thus linear with respect to the total number of entries in all three matrices, i.e.,  $O(pq + qr + pr)$ . More precisely, it takes time  $O(p \cdot r)$  to compute  $C'x^T$ ,  $O(q \cdot r)$  to compute  $Bx^T$ , and finally  $O(p \cdot q)$  to compute the product of  $A$  with  $Bx^T$ .

**Lemma 1.** *Let  $A$ ,  $B$ ,  $C'$  be three matrices of size  $p \times q$ ,  $q \times r$  and  $p \times r$ , respectively, over a ring. Suppose that  $C'$  is different from the matrix product  $C$  of  $A$  and  $B$  exactly in a single entry. We can identify this entry and correct it in time linear with respect to the total number of entries, i.e., in  $O(pq + qr + pr)$  time.*

## 4 Correcting a Matrix Product with at Most $k$ Errors

In this section, we shall repeatedly use a generalization of the deterministic version of Freivalds' test applied to detecting single erroneous entries in the previous section.

Let  $A$ ,  $B$  be two  $n \times n$  matrices, and let  $C'$  be their possibly faulty product matrix with at most  $k$  erroneous entries, over some ring. Let  $C^*$  and  $B^*$  denote matrices resulting from the same permutation of columns in the matrices  $C'$  and  $B$ .

Similarly as in the previous section, the generalized deterministic version of Freivalds' test verifies rows of  $C^*$ , but only for a selected set of consecutive columns of the matrix. Such a set of columns will be called a *strip*.

We shall check each strip of  $C^*$  independently for erroneous entries that occur in a single column of the strip. To do this, when we determine the vector  $v$  to be used in the coordinate-wise comparison of  $A(B^*v^T)$  with  $C^*v^T$ , we set the  $i$ -th coordinate of  $v$  to 1 if and only if the  $i$ -th column of the matrix  $C^*$  belongs to the strip we want to test. Otherwise, we set the coordinate to 0. (See Fig. 1.)

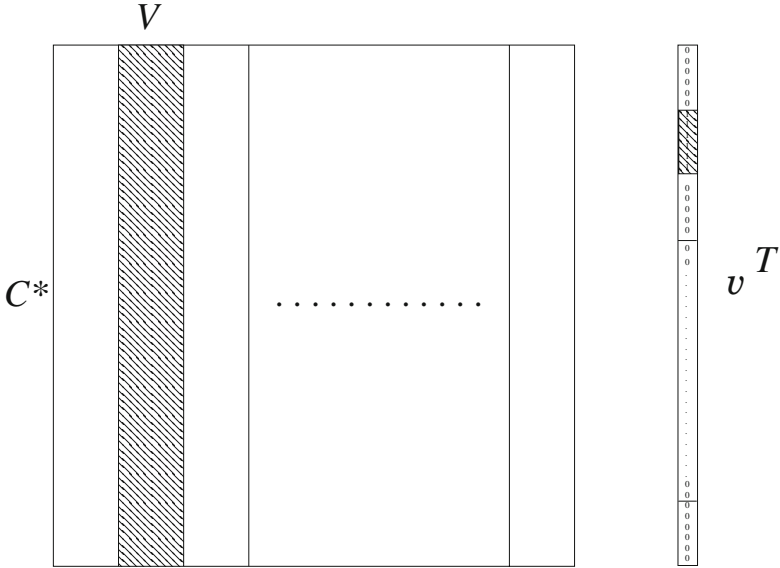
In this way, for each row in a strip, we can detect whether or not the strip row contains a single error. The time complexity for testing a whole strip in this way is  $O(n^2)$ , independently from the number of columns of the strip. If necessary, we can also correct a single row of a strip by recomputing all its entries in time proportional to  $n$  times the number of columns in the strip.

Our algorithm in this section relies also on the following number theoretical lemma.

**Lemma 2.** *Let  $P = \{i_1, \dots, i_l\}$  be a set of  $l$  different indices in  $\{1, \dots, n\}$ . There exists a constant  $c$  and for each  $i_m \in P$ , a prime  $p_m$  among the first  $cl \log n / \log \log n$  primes such that for  $i_q \in P \setminus \{i_m\}$ ,  $i_m \bmod p_m \neq i_q \bmod p_m$ .*

*Proof.* It follows from the Chinese remainder theorem, the density of primes and the fact that each index in  $P$  has  $O(\log n)$  bits that there is a constant  $b$  such that for each pair  $i_m, i_q$  of distinct indices in  $P$  there are at most  $b \log n / \log \log n$  primes  $p$  such that  $i_m \bmod p = i_q \bmod p$ . Consequently, for each  $i_m \in P$  there are at most  $b(l-1) \log n / \log \log n$  primes  $p$  for which there exists  $i_q \in P \setminus \{i_m\}$  such that  $i_q \bmod p = i_m \bmod p$ . Thus, it is sufficient to set the constant  $c$  to  $b$  in order to obtain the lemma.  $\square$

Given the generalized deterministic version of Freivalds' test and Lemma 2, the idea of our algorithm for correcting  $C'$  is simple, see Fig. 2.



**Fig. 1.** Illustration of using the vector  $v^T$  in order to “extract” the vertical strip  $V$  from the matrix  $C^*$

For each prime  $p$  among the first  $ck \log n / \log \log n$  primes, for  $j = 1, \dots, n$ , the  $j$ -th column is moved into a (vertical) strip corresponding to  $j \bmod p$ . Correspondingly, the columns of the matrix  $B$  are permuted.

Let  $B^*$  and  $C^*$  denote the resulting shuffled matrices.

Next, for each strip  $V$  of  $C^*$ , we set  $v$  to the vector in  $\{0, 1\}^n$  whose  $j$ -th coordinate is 1 if and only if the  $j$ -th column belongs to  $V$ . We compute and compare coordinate-wise the vectors  $A(B^*v^T)$  and  $C^*v^T$ . Note that for  $i = 1, \dots, n$ , if there is a single erroneous entry in the  $i$ -th row of  $V$  then the vectors  $A(B^*v^T)$ ,  $C^*v^T$  are different in this coordinate. Simply, the  $i$ -th coordinate of  $C^*v^T$  is just the sum of the entries in the  $i$ -th row of  $V$  while that coordinate of  $A(B^*v^T)$  is the sum of the entries in the  $i$ -th row of the vertical strip of the product of  $A$  and  $B^*$  corresponding to  $V$ .

It follows in particular that for each strip which contains only one erroneous column, we shall find all erroneous rows in the strip. Furthermore, we can correct all the erroneous entries in a detected erroneous row of the vertical strip  $V$  in  $O(n^2/p)$  time by computing  $O(n/p)$  dot products of rows of  $A$  and columns of  $B^*$ .

It follows from Lemma 2, that for each erroneous column in  $C'$ , there is such a prime  $p$  that the column is a single erroneous column in one of the aforementioned vertical strips of the shuffled matrix  $C^*$ . Hence, all the  $k$  errors can be localized and corrected.

**Algorithm 1**

*Input:* three  $n \times n$  matrices  $A$ ,  $B$ ,  $C'$  such that  $C'$  differs from the matrix product of  $A$  and  $B$  in at most  $k$  entries.

*Output:* the matrix product of  $A$  and  $B$ .

$L \leftarrow$  the set of the first  $ck \log n / \log \log n$  primes;

$C^* \leftarrow C'$ ;  $B^* \leftarrow B$ ;

**for** each prime  $p \in L$  **do**

1. **for**  $j = 1, \dots, n$  **do**
  - (a) Move the  $j$ -th column of  $C^*$  into the  $j \bmod p + 1$  strip of columns in  $C^*$ ;
  - (b) Correspondingly move the  $j$ -th column of  $B^*$  into the  $j \bmod p + 1$  strip of columns in  $B^*$ ;
2. **for** each strip  $V$  of  $C^*$  **do**
  - (a) Set  $v$  to the vector in  $\{0, 1\}^n$  whose  $j$ -th coordinate is 1 if and only if the  $j$ -th column of  $C^*$  belongs to  $V$ ;
  - (b) Compute the vectors  $A(B^*v^T)$  and  $C^*v^T$ ;
  - (c) **for** each coordinate  $i$  in which  $A(B^*v^T)$  and  $C^*v^T$  are different **do**
    - i. Compute the entries in the  $i$ -th row of the strip of  $A \times B^*$  corresponding to  $V$  and correct the  $i$ -th row of  $V$  in  $C$  appropriately.

Output  $C^*$ .

**Fig. 2.** A deterministic algorithm for correcting at most  $k$  errors

**Lemma 3.** *Let  $A$ ,  $B$ ,  $C'$  be three  $n \times n$  matrices over a ring. Suppose that  $C'$  is different from the matrix product  $C$  of  $A$  and  $B$  in at most  $k$  entries. Algorithm 1 identifies these erroneous entries and corrects them in  $\tilde{O}(k^2n^2)$  time.*

*Proof.* The correctness of Algorithm 1 (see Fig. 2) follows from the above discussion and Lemma 2.

Algorithm 1 iterates over  $ck \log n / \log \log n$  smallest primes. Since an upper bound on the  $i$ -th prime number is  $O(i \log i)$  for any  $i > 1$ , it follows that the largest prime considered by the algorithm has size  $O(ck \log n \log k)$ , and hence all these primes can be listed in  $O(c^2k^2 \log^2 n \log k)$  time.

For a given prime  $p$ , the algorithm tests  $p$  vertical strips  $V$  for the containment of rows with single errors by computing the vectors  $A(B^*v^T)$  and  $C^*v^T$ . It takes  $O(n^2p)$  time in total, for all these strips.

By the upper bounds on the number of considered primes and their size, it follows that the total time taken by the tests for all considered primes is  $O(c^2k^2n^2 \log^2 n \log k / \log \log n)$ .

The correction of an erroneous entry in a detected erroneous row in a vertical strip  $V$  takes  $O(n^2/p)$  time. Thus, the correction of the at most  $k$  erroneous entries in  $C^*$ , when the corresponding erroneous rows have been detected, takes total time  $O(kn^2)$ .

Hence, the upper time bound for the tests dominates the running time of the algorithm.  $\square$

In a practical implementation of the algorithm above, one can of course implement the shuffling of the columns without actually copying data from one column to another. For this purpose one could also define the strips in a different way, i.e., they do not need to consist of consecutive columns.

**Reducing the Time Bound to  $\tilde{O}(kn^2)$ .** In order to decrease the power of  $k$  in the upper bound of the time complexity from 2 to 1, we make the following observation. Consider any column  $i$  of  $C'$ . The number of erroneous entries in column  $i$  that are in rows that have at least  $\sqrt{k}$  erroneous entries is at most  $\sqrt{k}$ .

We start by applying Algorithm 1 with the difference that we only use the smallest  $c\sqrt{k} \log n / \log \log n$  primes. In this way all rows that have at most  $\sqrt{k}$  erroneous entries will be found in total  $\tilde{O}((\sqrt{k})^2 n^2)$  time, and will be fixed in  $O(n^2)$  time for each detected erroneous row. So the time complexity up to this stage is dominated by  $\tilde{O}(kn^2)$ .

Now, we let  $C''$  be the partially corrected matrix and we apply the same procedure but reversing the roles of columns and rows, i.e., we work with  $B^T A^T$  and  $C''^T$ . Since for any row of  $C''^T$ , all its erroneous entries that were in columns of  $C''^T$  with at most  $\sqrt{k}$  errors were already corrected, now by the observation, the number of erroneous entries in any row of  $C''^T$  is at most  $\sqrt{k}$ . Thus Algorithm 1 will now find all remaining erroneous rows in time  $\tilde{O}(kn^2)$  and we can correct them in additional time  $O(kn^2)$ . Hence we obtain the following theorem:

**Theorem 1.** *Let  $A$ ,  $B$ ,  $C'$  be three  $n \times n$  matrices over a ring. Suppose that  $C'$  is different from the matrix product  $C$  of  $A$  and  $B$  in at most  $k$  entries. We can identify these erroneous entries and correct them in  $\tilde{O}(kn^2)$  time.*

**Few Random Bits Help.** We can decrease the power of  $k$  in the upper bound of Theorem 1 from 1 to 0.5 by using  $O(\log^2 k + \log k \log \log n)$  random bits as follows and assuming that the exact number  $k$  of erroneous entries in  $C'$  is known. (The removal of this assumption will be discussed later.) The idea is that instead of testing systematically a sequence of primes, we start by producing four times as many primes and then choose randomly among them in order to produce the strips.

We call a faulty entry in  $C'$  1-detectable if it lies in a row or column of  $C'$  with at most  $2\sqrt{k}$  erroneous entries. From this definition it follows that most faulty entries are 1-detectable. More specifically, we call an entry in  $C'$  1-row-detectable, respectively 1-column-detectable, if it lies in a row, respectively column, with at most  $2\sqrt{k}$  erroneous entries.

We will aim at detecting first a constant fraction of the 1-row-detectable (false) entries, and then a constant fraction of the 1-column-detectable entries. For this purpose we start by producing, in a preprocessing phase, the smallest  $4c\sqrt{k} \log n / \log \log n$  primes (i.e., four times as many primes as we did in the deterministic algorithm of Theorem 1).

To detect sufficiently many 1-row-detectable entries we run one iteration of Algorithm 1, with the difference that we use a prime chosen randomly among

the produced  $4c\sqrt{k} \log n / \log \log n$  smallest primes. In this way, for each 1-row-detectable entry there is at least a probability  $1/2$  that it will be detected.

Then we repeat once more this procedure but reversing the role of columns and rows, i.e., by working with  $B^T A^T$  and  $C'^T$ . In this way for each 1-column-detectable entry there is at least a probability  $1/2$  that it will be detected.

In this way, now each 1-detectable entry has been detected with probability at least  $1/2$ . By correcting all these detected entries, we thus reduce the total number of remaining false entries by an expected constant fraction.

Thus we can set  $k$  to the remaining number of false entries and start over again with the resulting, partially corrected matrix  $C'$ . We repeat in this way until all erroneous entries are corrected.

The expected time bound for the tests and corrections incurred by the first selected primes dominate the overall expected time complexity. Note that the bound is solely  $O(c\sqrt{kn}^2 \log n \log k)$ .

The number of random bits needed to select such a random prime is  $O(\log k + \log \log n)$ . The overall number of random bits, if we proceed in this way and use fresh random bits for every new selection of a prime number, has to be multiplied by the expected number of the  $O(\log k)$  iterations of the algorithm. Thus, it becomes  $O(\log^2 k + \log k \log \log n)$ .

Hence, we obtain the following slightly randomized version of Theorem 1.

**Theorem 2.** *Let  $A$ ,  $B$ ,  $C'$  be three  $n \times n$  matrices over a ring. Suppose that  $C'$  is different from the matrix product  $C$  of  $A$  and  $B$  in exactly  $k$  entries. There is a randomized algorithm that identifies these erroneous entries and corrects them in  $\tilde{O}(\sqrt{kn}^2)$  expected time using  $O(\log^2 k + \log k \log \log n)$  random bits.*

If the number  $k$  or erroneous entries is not known, then our slightly randomized method can be adapted in order to estimate the number of erroneous columns and rows. Since similar issues arise in connection to another randomized approach presented in the next chapter, we postpone this discussion to Final Remarks.

## 5 A Faster Randomized Approach

In this section, similarly as in the previous one, we shall repeatedly apply a version of Freivalds' test to (vertical) strips of the possibly erroneous matrix product  $C'$  of two  $n \times n$  matrices  $A$  and  $B$ . However, in contrast with the previous section, the test is randomized. It is just a restriction of Freivalds' original randomized algorithm [7] to a strip that detects each erroneous row of a strip with probability at least  $1/2$  even if a row contains more than one erroneous entry.

More precisely, the vector  $v$  used to test a strip of  $C'$  by comparing  $A(Bv^T)$  with  $C'v^T$  is set as follows. For  $j = 1, \dots, n$ , the  $j$ -th coordinate of  $v$  is set to 1 independently with probability  $1/2$  if and only if the  $j$ -th column of  $C'$  belongs to the strip we want to test, otherwise the coordinate is set to 0. In this way, for each row in the strip, the test detects whether or not the strip row contains an

erroneous entry with probability at least  $1/2$ , even if the row contains more than one erroneous entry. The test for a whole strip takes  $O(n^2)$  time, independently from the number of columns of the strip.

Using the aforementioned strip test, we shall prove the following theorem.

**Theorem 3.** *Let  $A$ ,  $B$  and  $C'$  be three  $n \times n$  matrices over a ring. Suppose that  $C'$  is different from the matrix product  $C$  of  $A$  and  $B$  in  $k$  entries. There is a randomized algorithm that transforms  $C'$  into the product  $A \times B$  in  $O(\sqrt{k} \cdot n^2 \cdot \log n)$  time almost surely without assuming any prior knowledge of  $k$ .*

*Proof.* Let us assume for the moment that  $k$  is known in advance (this assumption will be removed later). Our algorithm (see Algorithm 2 in Fig. 2) will successively correct the erroneous entries of  $C'$  until  $C'$  will become equal to  $A \times B$ . For easier description, let us also assume that  $\sqrt{k}$  is an integer, and that  $n$  is a multiple of  $\sqrt{k}$ .

We consider a partition of the columns of  $C'$  into  $\sqrt{k}$  strips of equal size, i.e., consecutive groups of  $n/\sqrt{k}$  columns of  $C'$ . We treat each such strip separately and independently. For each strip, we apply our version of Freivalds' test  $O(\log n)$  times. In this way, we can identify almost surely which rows of the tested strip contain at least one error. (Recall that for each iteration and for each strip row, the chance of detecting an error, if it exists, is at least  $1/2$ .) Finally, for each erroneous strip row, we compute the correct values for each one of its  $n/\sqrt{k}$  entries.

### Algorithm 2

*Input:* three  $n \times n$  matrices  $A$ ,  $B$ ,  $C'$  such that  $C'$  differs from the matrix product of  $A$  and  $B$  in at most  $k$  entries.

*Output:* the matrix product of  $A$  and  $B$ , almost surely.

**for**  $i = 1, \dots, \lceil \sqrt{k} \rceil$  **do**

1. Run the strip restriction of Freivalds' algorithm  $c \cdot \log n$  times on the  $i$ -th (vertical) strip of  $C'$ ;
2. For each erroneous strip row found in the  $i$ -th (vertical) strip of  $C'$ , compute each entry of this strip row and update  $C'$  accordingly;

Output  $C'$ .

**Fig. 3.** A randomized algorithm for correcting at most  $k$  errors

In each iteration of the test in Step 1 in the algorithm, each erroneous row in the strip will be detected with a probability at least  $1/2$ . Hence, for a sufficiently large constant  $c$  (e.g.,  $c=3$ ) all erroneous rows will be detected almost surely within  $c \cdot \log n$  iterations. If we use the straightforward method in order to compute the correct values of an erroneous strip row, then it will take  $O(n)$  time per entry. Since each strip row contains  $n/\sqrt{k}$  entries, the time taken by a strip row becomes  $O(n^2/\sqrt{k})$ . Since there are at most  $k$  erroneous strip rows, the total

time for correcting all the erroneous strip rows in all strips is  $O(\sqrt{k} \cdot n^2)$ . Hence, the total time complexity is dominated by applying the strip tests,  $O(\log n)$  times for each one of the strips. This yields an upper time bound of  $O(\sqrt{k} \cdot n^2 \cdot \log n)$ .

In Algorithm 2, if we use, instead of the correct number  $k$  of erroneous entries, a guessed number  $k'$  which is larger than  $k$ , then the time complexity becomes  $O(\sqrt{k'} \cdot n^2 \cdot \log n)$ . This would be asymptotically fine as long as  $k'$  is within a constant factor of  $k$ . On the other hand, if we guess  $k'$  which is much smaller than  $k$ , then the length of each erroneous strip row may become too large. For this reason, first we have to find an appropriate size for the strips to be used by our algorithm. For this purpose, we start by setting  $k'$  to a small constant, e.g., to 4, and then we multiply our guess by 4, until we reach a good balance. More precisely, for each such guessed  $k'$ , without correcting any errors, we consider a partition of the matrix  $C'$  into  $\sqrt{k'}$  strips, and apply our test to each strip. As soon as we discover more than  $k'$  erroneous strip rows we break the procedure without correcting any errors, and we start over with a four times larger guess  $k'$ .

The aforementioned method of guessing  $k'$  may result in at most  $O(\log k)$  wrong guesses until we achieve a good guess. Since we multiply our guess every time with 4, we obtain a geometric progression of the estimated costs of subsequent trials. In this way, the upper bound on the asymptotic complexity of the whole algorithm is dominated by that of the final step. In this step, we test each strip  $c \cdot \log n$  times in order to detect almost surely all erroneous strip rows.

Note that when the number of erroneous entries is at most four then our algorithm will keep its first guess, i.e.,  $k' = 4$ , and so the number of strips will be (and remain) 2. Hence, it will correct at most 4 erroneous rows in total time  $O(n^2)$ . So, we can focus on the case when  $k > 4$ . With respect to our current guess  $k'$ , the number of detected erroneous rows lies thus almost surely between  $k'/4$  and  $k'$ . Since each such an erroneous row contains  $n/\sqrt{k'}$  entries, it can be recomputed in  $O(n^2/\sqrt{k'})$  time. Consequently, the total time complexity of correcting all the at most  $k'$  erroneous rows becomes  $O(k' \cdot n^2 / \sqrt{k'}) = O(\sqrt{k'} \cdot n^2)$ . Since  $k' \leq 4k$  holds, the theorem follows.  $\square$

Algorithm 2 in the proof of Theorem 3 can be modified in order to achieve an expected time bound of  $O(\sqrt{k} \cdot n^2)$  for correcting all errors, if  $k$  is known in advance. (We discuss the removal of this assumption in Final Remarks.) Instead of applying the strip restriction of Freivalds' algorithm  $c \cdot \log n$  times for each strip, we apply it only once for each strip and correct all erroneous rows which we detect. By counting how many errors we have corrected, we compute how many errors remain. Then we recurse in the same way on the partially corrected matrix  $C'$  using as a parameter this new number of errors which remain to be corrected.

During each iteration of the algorithm, each remaining error in  $C'$  will be detected and corrected with probability at least  $1/2$ . Thus, the expected number of remaining errors will be halved after each iteration. Consequently, we obtain a geometric progression on the expected time complexity of each iteration, and so the total expected time complexity is dominated by the time taken by the first iteration, which is  $O(\sqrt{k} \cdot n^2)$ . Thus we obtain the following theorem.



**Theorem 4.** *Let  $A, B, C'$  be three  $n \times n$  matrices over a ring. Suppose that  $C'$  is different from the matrix product  $C$  of  $A$  and  $B$  in exactly  $k$  entries. There is a randomized algorithm that identifies these erroneous entries and corrects them in  $O(\sqrt{k} \cdot n^2)$  expected time.*

## 6 Final Remarks

The algorithm used for Theorem 4 can be adapted for the case when the number  $k$  of errors is unknown, by making guesses  $k'$  of the form  $4^l$ , similarly to the proof of Theorem 3, starting with  $k' = 4$ . For each new guess  $k'$ , we divide the matrix  $C'$  into  $\sqrt{k'}$  strips and apply the strip-restricted variant of Freivalds' algorithm only once for each strip, counting the number of detected erroneous strip rows, without performing any corrections. If the number of detected erroneous strip rows is greater than  $k'$ , we break the procedure and start over with a four times larger guess. Otherwise, we correct all errors in the detected erroneous strips, and start over the algorithm with the partially corrected matrix  $C'$ . However, as a final phase we may have to perform  $O(\log n)$  additional iterations to be sufficiently sure that no errors remain.

A similar approach can also be used for refining the slightly randomized method of Theorem 2 when the number of errors  $k$  is not known in advance. However, if there is no knowledge at all concerning the number of errors, it may be difficult to handle the case when no errors are detected: does this happen because there are no errors at all, or because there are too many errors and we chose a random prime from a too small range, thus failing to isolate 1-detectable false entries? For this reason, if there is no known useful upper bound on the remaining number of errors, and we do not detect any errors during a series of iterations, we may have to resort to some of the known algorithms which test whether there are any errors at all [2, 9, 12]. All such known algorithms running in time  $O(n^2)$  may need a logarithmic number of random bits, so if  $k$  is very small then this may be asymptotically larger than the low number of random bits stated in Theorem 2.

Finally, observe that any substantial improvement of our  $\tilde{O}(\sqrt{k}n^2)$  bound by a combinatorial method seems to be very hard to achieve. Simply, it would lead to a substantially subcubic combinatorial algorithm for Boolean matrix multiplication which would be a breakthrough [15].

**Acknowledgments.** We thank the anonymous referees for helping us to improve a previous version of this paper.

## References

1. Buhrman, H., Spalek, R.: Quantum Verification of Matrix Products. In: Proc. ACM-SIAM SODA, pp. 880–889 (2006)
2. Chen, Z.-Z., Kao, M.-Y.: Reducing Randomness via Irrational Numbers. In: Proc. ACM STOC, pp. 200–209 (1997)

3. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. *J. of Symbolic Computation* **9**, 251–280 (1990)
4. De Bonis, A., Gasieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. *SIAM Journal on Computing* **34**(5), 1253–1270 (2005)
5. Ding, C., Karlsson, C., Liu, H., Davies, T., Chen, Z.: Matrix Multiplication on GPUs with On-Line Fault Tolerance. In: *Proc. of the 9th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2011)*, Busan, Korea, May 26–28 (2011)
6. Du, D.Z., Hwang, F.K.: *Combinatorial Group Testing and its Applications* World Scientific Publishing, NJ (1993)
7. Freivalds, R.: Probabilistic Machines Can Use Less Running Time. *IFIP Congress* pp. 839–842 (1977)
8. Le Gall, F.: Powers of Tensors and Fast Matrix Multiplication. In: *Proc. 39th International Symposium on Symbolic and Algebraic Computation, (ISSAC 2014)*, pp. 296–303 (2014)
9. Kimbrel, T., Sinha, R.K.: A probabilistic algorithm for verifying matrix products using  $O(n^2)$  time and  $\log_2 n + O(1)$  random bits. *Information Processing Letters* **45**, 107–119 (1993)
10. Korec, I., Wiedermann, J.: Deterministic Verification of Integer Matrix Multiplication in Quadratic Time. In: Geffert, V., Preneel, B., Rován, B., Štuller, J., Tjoa, A.M. (eds.) *SOFSEM 2014*. LNCS, vol. 8327, pp. 375–382. Springer, Heidelberg (2014)
11. McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. *Computer Science Review* **5**(2), 119–161 (2011)
12. Naor, J., Naor, M.: Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. Comput.* **22**(4), 838–856 (1993)
13. Strassen, V.: Gaussian elimination is not optimal. *Numerische Mathematik* **13**, 354–356 (1969)
14. Vassilevska Williams, V.: Multiplying matrices faster than Coppersmith-Winograd. In: *Proc. ACM STOC*, pp. 887–898 (2012)
15. Vassilevska Williams, V., Williams, R.: Subcubic Equivalences between Path, Matrix and Triangle Problems. In: *Proc. IEEE FOCS 2010*, pp. 645–654 (2010)
16. Wu, P., Ding, C., Chen, L., Gao, F., Davies, T., Karlsson, C., Chen, Z.: Fault Tolerant Matrix-Matrix Multiplication: Correcting Soft Errors On-Line. In: *Proc. of the 2011 Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA) held in conjunction with the 24th IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC 2011)* (2011)
17. Wu, P., Ding, C., Chen, L., Gao, F., Davies, T., Karlsson, C., Chen, Z.: On-Line Soft Error Correction in Matrix-Matrix Multiplication. *Journal of Computational Science* **4**(6), 465–472 (2013)

# 3D Rectangulations and Geometric Matrix Multiplication

Peter Floderus<sup>1</sup>, Jesper Jansson<sup>2</sup>, Christos Levkopoulos<sup>3</sup>,  
Andrzej Lingas<sup>3</sup> (✉), and Dzmitry Sledneu<sup>1</sup>

<sup>1</sup> Centre for Mathematical Sciences, Lund University, 22100 Lund, Sweden  
`{pflo,Dzmitry}@maths.lth.se`

<sup>2</sup> Laboratory of Mathematical Bioinformatics, Institute for Chemical Research,  
Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan  
`jj@kuicr.kyoto-u.ac.jp`

<sup>3</sup> Department of Computer Science, Lund University, 22100 Lund, Sweden  
`{Christos.Levkopoulos,Andrzej.Lingas}@cs.lth.se`

**Abstract.** The problem of partitioning an input rectilinear polyhedron  $P$  into a minimum number of 3D rectangles is known to be NP-hard. We first develop a 4-approximation algorithm for the special case in which  $P$  is a 3D histogram. It runs in  $O(m \log m)$  time, where  $m$  is the number of corners in  $P$ . We then apply it to compute the arithmetic matrix product of two  $n \times n$  matrices  $A$  and  $B$  with nonnegative integer entries, yielding a method for computing  $A \times B$  in  $\tilde{O}(n^2 + \min\{r_A r_B, n \min\{r_A, r_B\}\})$  time, where  $\tilde{O}$  suppresses polylogarithmic (in  $n$ ) factors and where  $r_A$  and  $r_B$  denote the minimum number of 3D rectangles into which the 3D histograms induced by  $A$  and  $B$  can be partitioned, respectively.

**Keywords:** Geometric decompositions · Minimum number rectangulation · Polyhedron · Matrix multiplication · Time complexity

## 1 Introduction

This paper considers two intriguing and at a first glance unrelated problems.

The first problem lies at the heart of three-dimensional computational geometry. It belongs to the class of *polyhedron decomposition* problems, whose applications range from data compression and database systems to pattern recognition, image processing, and computer graphics [7, 13]. The problem is to partition a given rectilinear polyhedron into a minimum number of 3D rectangles. Dielissen and Kaldewai have shown this problem to be NP-hard [4]. In contrast, the problem of partitioning a rectilinear (planar) polygonal region into a minimum number of 2D rectangles admits a polynomial-time solution [7, 10]. Formally, the NP-hardness proof by [4] is for polyhedra with holes, but the authors remark that the proof should also work for simple polyhedra. To the best of our knowledge,

---

Jesper Jansson: Funded by The Hakubi Project at Kyoto University.

Christos Levkopoulos: Research supported in part by Swedish Research Council grant 621-2011-6179.

no non-trivial approximation factors for minimum rectangular partition of simple rectilinear polyhedra are known, even in restricted non-trivial cases such as that of a 3D histogram (a natural generalization of a planar histogram, see Section 2).

The second problem we consider is that of multiplying two  $n \times n$  matrices. There exist fast algorithms that do so in substantially subcubic time, e.g., a recent one due to Le Gall runs in  $O(n^{2.3728639})$  time [8], but they suffer from very large overheads. On the positive side, input matrices in real world applications often belong to quite restricted matrix classes, so a natural approach is to design faster algorithms for such special cases. Indeed, efficient algorithms for *sparse* matrix multiplication have been known for long time. In the Boolean case, despite considerable efforts by the algorithms community, the fastest known combinatorial algorithms for Boolean  $n \times n$  matrix multiplication barely run in subcubic time (in  $O(n^3(\log \log n)^2/(\log n)^{9/4})$  time [1], to be precise), but much faster algorithms for Boolean matrix product for restricted classes of Boolean matrices have been developed [3, 5, 9]. For example, when at least one of the input Boolean matrices admits an exact covering of its ones by a relatively small number of rectangular submatrices, the Boolean matrix product can be computed efficiently [9]; similarly, if the rows of the first input Boolean matrix or the columns of the second input Boolean matrix can be represented by a relatively cheap minimum cost spanning tree in the Hamming metric (or its generalization to include blocks of zeros or ones) then the Boolean matrix product can be computed efficiently by a randomized combinatorial algorithm [3, 5].

Our first contribution is an  $O(m \log m)$ -time, 4-approximation algorithm for computing a minimum 3D rectangular partition of an input 3D histogram with  $m$  corners. It works by projecting the input histogram onto the base plane, partitioning the resulting planar straight-line graph into a number of 2D rectangles not exceeding its number of vertices, and transforming the resulting 2D rectangles into 3D rectangles of appropriate height. Importantly, the known algorithms for minimum partition of a rectilinear polygon with holes into 2D rectangles [7, 10] do not yield the aforementioned upper bound on the number of rectangles in the more general case of planar straight-line graphs.

Our second contribution is a new technique for multiplying two matrices with nonnegative integer entries. We interpret the matrices as 3D histograms and decompose them into blocks that can be efficiently manipulated in a pairwise manner using the interval tree data structure. Let  $A$  and  $B$  be two  $n \times n$  matrices with nonnegative integer entries, and let  $r_A$  and  $r_B$  denote the minimum number of 3D rectangles into which the 3D histograms induced by  $A$  and  $B$  can be partitioned. By applying our 4-approximation algorithm above, we can compute  $A \times B$  in  $\tilde{O}(n^2 + r_A r_B)$  time, where  $\tilde{O}$  suppresses polylogarithmic (in  $n$ ) factors. Furthermore, by using another idea of slicing the histogram of  $A$  (or  $B$ ) into parts corresponding to rows of  $A$  (or columns of  $B$ ) and measuring the cost of transforming a slice into a consecutive one, we obtain an upper bound of  $\tilde{O}(n^2 + n \min\{r_A, r_B\})$ . We also give a generalization of the latter upper bound in terms of the minimum cost of a spanning tree of the slices, where the distance between a pair of slices corresponds to the cost of transforming one slice into the other.

**Organization:** Section 2 presents our 4-approximation algorithm for a partition of a 3D histogram into a minimum number of 3D rectangles. Section 3 presents our algorithms for the arithmetic matrix product. Section 4 concludes with some final remarks.

## 2 3D Histograms and Their Rectangular Partitions

A 2D histogram is a polygon with an edge  $e$ , which we call the *base* of the histogram, having the following property: for every point  $p$  in the interior of histogram, there is a (unique) line segment perpendicular to  $e$ , connecting  $p$  to  $e$  and lying totally in the interior of the histogram. In this paper, we consider orthogonal histograms only. For simplicity, we consider the base of a histogram as being horizontal, and all other edges of the histogram lying above the base. In this way, a 2D histogram can also be thought of as the union of rectangles standing on the base of the histogram.

A *3D histogram* is a natural generalization of a 2D histogram. To define a 3D histogram, we need the concept of the “base plane”, which for simplicity we define as the horizontal plane containing two of the axes in the Euclidean space. A 3D histogram can then be thought of as the union of rectilinear 3D rectangles, standing on the base plane. The *base of the histogram* is the union of the lower faces (also called *bases*) of all these rectangles.

**Definition 1.** *A 3D histogram is a union of a finite set  $C$  of rectilinear 3D rectangles such that: (i) each element in  $C$  has a face on the horizontal base plane; and (ii) all elements in  $C$  are located above the base plane.*

(In the literature, what we call a 3D histogram is sometimes termed a 2D histogram or a 1D histogram when used to summarize 2D or 1D data, respectively [12].)

By a *rectangular partition* of 3D histogram  $P$ , we mean a rectilinear partition of  $P$  into 3D rectangles. In Section 2.2 below, we consider the problem of finding a rectangular partition of a given 3D histogram  $P$  into as few 3D rectangles as possible. We present a 4-approximation algorithm for this problem with time complexity  $O(m \log m)$ , where  $m$  denotes the number of vertices in  $P$ . The algorithm partitions  $P$  into less than  $m'$  3D rectangles, where  $m'$  is the number of vertices in the vertical projection of  $P$  (i.e.,  $m' < m$ ), by applying a subroutine described in Section 2.1 that partitions any rectilinear planar straight-line graph (PSLG) with  $m'$  vertices into less than  $m'$  2D rectangles. Finally, the approximation factor is derived by observing that any rectangular partition of  $P$  must contain at least  $m'/4$  3D rectangles.

### 2.1 Partitioning a Rectilinear PSLG into 2D Rectangles

The problem of partitioning a rectilinear polygon into rectangles in two dimensions has been well studied in the literature [7, 10]. An optimal solution for this problem can be computed in polynomial time [7, 10]. However, to use the result in 3D, we need a bound on the number of produced rectangles, expressed in

terms of the number of vertices. Therefore, it is not so crucial for our purposes to compute an optimal solution for the 2-dimensional problem, but instead, we need to partition planar straight-line graphs (PSLGs) into at most  $m'$  rectangles, where  $m'$  denotes the number of vertices in the input PSLG. We will show that a simple algorithm suffices to obtain this bound.

Since this subsection considers 2D only, we use the term “horizontal” for line segments parallel to the  $X$ -axis. By “vertical” lines, we mean lines or line segments parallel to the  $Y$ -axis. Each vertex in the planar graphs in our application has degree 2, 3, or 4.

**Definition 2.** A planar straight-line graph (PSLG)  $PG = (V, E)$ , as used in this paper, is a planar graph where every vertex has an  $x$ - and a  $y$ -coordinate. Each edge is drawn as a straight line segment, all edges meet at right angles, and each vertex has degree 2, 3, or 4. A rectangular partition of  $PG$  is a partition  $R = (V \cup V_R, E \cup E_R)$  that adds edges and vertices to  $PG$  so that  $R$  is still a PSLG while every face in  $R$  is a rectangle.

Given a PSLG  $PG$ , we denote  $m' = |V|$ . We say that a vertex  $v$  of  $PG$  is *concave* if it has degree 2, its two adjacent edges are perpendicular to each other, and the corner at  $v$  which is of 270 degrees does not lie in the outer, infinite face of  $PG$ . Any vertex which is not concave is called *convex*.

We use a sweep line approach to generate a partition into less than  $m'$  rectangles. We perform a horizontal sweep with a vertical sweep line [2], using the vertices of  $PG$  as event points. Whenever the sweep line reaches a concave vertex  $v$ , we insert into the graph  $PG$  a vertical line segment  $s$  connecting  $v$  to the closest edge of PSLG upwards or downwards, thus canceling the concavity at  $v$  and transforming  $v$  into a convex vertex of degree 3. Hence, if there was already an edge of  $PG$  below  $v$ , then the new segment  $s$  is inserted above  $v$ , otherwise it is inserted below  $v$ . To preserve the property that the resulting graph is still a PSLG, the other endpoint of  $s$  may have to become a new vertex of the PSLG. This is a standard procedure for trapezoidation; see, e.g., [2] for more details. After the sweep is complete, all concave vertices have been eliminated. (Remark: In a special case it may happen that two concave vertices with the same  $x$ -coordinate are connected by a single vertical segment that is disjoint from the rest of the input PSLG. In this case, the plane sweep algorithm will produce this segment. Thus, no two segments produced by the algorithm overlap or touch each other.)

The correctness of the algorithm is easy to see: it eliminates all concave corners of  $PG$  by adding vertical line segments. Hence, in the resulting PSLG, each face, except for the outer face, is a rectangle. The running time of this algorithm is dominated by the cost of the plane sweep, which is  $O(m' \log m')$  according to well-known methods in computational geometry; see, e.g., [2].

We need to relate the number of vertices in the input PSLG to the number of 2D rectangles. This is done in the following lemma:

**Lemma 1.** Any PSLG  $PG = (V, E)$  with  $|V| = m'$  and minimum vertex degree 2 can be partitioned into  $b$  rectangles with  $b < m'$  using  $O(m' \log m')$  time.

*Proof.* Let  $R$  denote the set of rectangles in the rectangular partition produced by the plane sweep algorithm described above. We use a “charging scheme” to prove the stated inequality. The charging scheme starts by giving each vertex  $v \in V$  four tokens; thus, a total of  $4m'$  tokens are used. Each vertex  $v$  then distributes its tokens in a certain way to the rectangles in  $R$  that are adjacent to  $v$ . We will show that every rectangle in  $R$  receives at least four tokens. Since we started by giving a total of  $4m'$  tokens to the vertices, this will prove that there exist at most  $m'$  rectangles, and thus  $b \leq m'$ . Moreover, vertices adjacent to the outer face do not give away more than three tokens. We will thus obtain the strict inequality  $b < m'$ .

Now, we describe the details of the charging scheme. (More explanations and illustrating figures are included in the full version.) Let  $v$  be any vertex of  $V$ . The vertex  $v$  gives one token to each rectangle  $r$  in  $R$  which in any way is adjacent to it, with one exception. The exception occurs when  $v$  is a concave vertex; then,  $v$  is partitioned by a vertical segment  $e_r$  added by the algorithm. This segment partitions the three quadrants at the concave corner around the vertex so that one rectangle occupies one quadrant and one occupies the two others. Then  $v$  distributes two tokens to the new rectangle occupying only one quadrant, which therefore has a corner at  $v$ , and only one token to each one of the other rectangles of  $R$  adjacent to  $v$ .

We now show that each rectangle receives at least four tokens. Let  $r$  be any rectangle in  $R$ . First note that each vertical segment added by the algorithm has at least one endpoint at a vertex in  $V$ . Moreover, for any rectangle  $r$  in  $R$ , each of the vertical sides of  $r$  includes at least one vertex of  $V$ . Therefore, each rectangle is adjacent to at least two vertices of  $V$ . We distinguish three cases, depending on the number of vertices of  $V$  adjacent to  $r$ . Observe that the adjacencies are not necessarily at the corners of  $r$ .

- Case 1:  $r$  is adjacent to at least four vertices of  $V$ . Since  $r$  will receive at least one token from each of them we are done.
- Case 2:  $r$  is adjacent to precisely three vertices of  $V$ . Then at one of the vertical sides of  $r$  there is only one vertex of  $V$ . Moreover, this vertex  $v$  must be at a corner of  $r$  and fulfills the criteria for giving two tokens to  $r$ . The remaining two adjacent vertices of  $V$  give at least one token each, so we are done.
- Case 3:  $r$  is adjacent to precisely two vertices of  $V$ . This must mean that both vertical sides of  $r$  are segments added by the algorithm, and that one of the endpoints of each of these sides is a vertex of  $V$  at a corner of  $r$ . This corresponds to the condition for receiving two tokens mentioned earlier. So in total,  $r$  receives four tokens from the two corners, and we are done.  $\square$

## 2.2 Partitioning a 3D Histogram into 3D Rectangles

We now explain how to obtain the projected PSLG from the 3D histogram  $P$  and how to use the rectangular partition of this PSLG to yield a good partition into 3D rectangles.

**Definition 3.** *The planar projection  $PP$  is an orthogonal projection of the input 3D histogram  $P$  along the “down” direction onto the base plane in Definition 1.*

We can interpret  $PP$  as a PSLG where each corner and each subdividing point on a line segment corresponds to a vertex. The edges naturally correlate to the connecting line segments between vertices. Each vertex in  $PP$  is the vertical projection of at least two vertices of  $P$ . Two edges of the 3D histogram may partially overlap in the 2D projection, but the edges in the 2D projection are considered as non-overlapping. Thus, an edge of the 3D histogram may split into several edges in the 2D projection, since vertices should only appear as endpoints of edges.

*Remark 1.* Every vertex in  $PP$  must have at least two neighbors. This follows from the fact that each vertex of  $P$  (and of any orthogonal polyhedron) has at least two incident horizontal edges. It may happen that some vertex of  $PP$  is the vertical projection of up to four vertices of  $P$ , so those four vertices of  $P$  may have a total of eight neighbors in  $P$ . But since  $PP$  is an orthogonal PSLG, no vertex of  $PP$  has more than four neighbors.

Now we are ready to show the main theorem of this section.

**Theorem 1.** *For any 3D histogram  $P$  with  $m$  corners, a 4-approximation  $R$  of a partition of  $P$  into as few 3D rectangles as possible can be computed in  $O(m \log m)$  time.*

*Proof.* We use the projection in Definition 3, let  $PG = PP$ , and apply Lemma 1 to compute a planar partition  $R'$ . The final 3D partition  $R$  is obtained from  $R'$  by reversing the projection so that each 2D rectangle corresponds to the top of a 3D rectangle in  $R$ .

To analyze the approximation factor, denote the number of 3D rectangles in an optimal solution  $R^*$  by  $OPT$  and the number of 3D rectangles produced by the algorithm described above by  $b$ . We denote by  $m'$  the number of vertices in  $PP$ . By Lemma 1, we have  $b < m'$  since each 2D rectangle corresponds to one 3D rectangle. Every vertex of  $P$  must be adjacent to at least one vertical edge of a 3D rectangle in  $R^*$ . Hence, each vertex in  $PP$  has to be at a corner of the vertical projection of at least one 3D rectangle in  $R^*$  onto the base plane. Since each 3D rectangle in  $R^*$  only has 4 vertical edges, its vertical projection can be adjacent to at most 4 vertices of  $PP$ . It follows that  $m' \leq 4OPT$  and  $b < m' \leq 4OPT$ .

Since the projection can be obtained by contracting each corner in  $P$  and all of its vertical neighbors into one vertex, the projection can be implemented in  $O(m)$  time. Thus, the  $O(m \log m)$ -term from Lemma 1 will dominate the time complexity.  $\square$

### 3 Geometric Algorithms for Arithmetic Matrix Product

#### 3.1 Geometric Data Structures and Notation

Our algorithms for arithmetic matrix multiplication use some data structures for interval and rectangle intersection. An *interval tree* is a leaf-oriented binary



search tree that supports intersection queries for a set  $Q$  of closed intervals on the real line as follows:

**Fact 1** [11]. *Suppose that the left endpoints of the intervals in a set  $Q$  belong to a subset  $\mathcal{U}$  of real numbers of size  $l$  and  $|Q| = q$ . An interval tree  $T$  of depth  $O(\log l)$  for  $Q$  can be constructed in  $O(l + q \log lq)$  time using  $O(l + q)$  space. The insertion or deletion of an interval with left endpoint in  $\mathcal{U}$  into  $T$  takes  $O(\log l + \log q)$  time. The intersection query is supported by  $T$  in  $O(\log l + r)$  time, where  $r$  is the number of reported intervals.*

*Remark 2.* The interval tree of Fact 1 ([11]) can easily be generalized to the weighted case, where with an interval to insert or delete an integer weight is associated. It can be done by maintaining in each node of the interval tree the sum of weights of intervals whose fragments it represents. In effect, the generalized interval insertions or deletions as well the intersection query have the same time complexity as those in Fact 1. Moreover, the generalized interval tree supports a weight intersection query asking for the total weight of the intervals containing the query point in  $O(\log l + \log q)$  time.

We use the following data structure, easily obtained by computing all prefix sums:

**Fact 2.** *For a sequence of integers  $a_1, a_2, \dots, a_n$ , one can construct a data structure that supports a query asking for reporting the sum  $\sum_{k=i}^j a_k$  for  $1 \leq i \leq j \leq n$  in  $O(1)$  time. The construction takes  $O(n)$  time.*

In the rest of the paper,  $A$  and  $B$  denote two  $n \times n$  matrices with nonnegative integer entries, and  $C$  stands for their matrix product. We also need the following concepts:

1. For an  $n \times n$  matrix  $D$  with nonnegative integer entries, consider the  $[0, n] \times [0, n]$  integer grid whose unit cells are in one-to-one correspondence with the entries of  $D$ . The grid cell between the horizontal lines  $i - 1$  and  $i$  (counting from the top) and vertical lines  $j - 1$  and  $j$  (counting from the left) corresponds to  $D_{i,j}$  (see Fig. 1a). Then,  $his(D)$  stands for the 3D histogram whose base consists of all unit cells of the  $[0, n] \times [0, n]$  integer grid corresponding to positive entries of  $D$  and whose height over the cell corresponding to  $D_{i,j}$  is the value of  $D_{i,j}$  (see Fig. 1b).
2. For the  $n \times n$  matrix  $D$ , nonnegative integers  $1 \leq i_1 \leq i_2 \leq n$ ,  $1 \leq k_1 \leq k_2 \leq n$ , and  $h_1, h_2$ , where  $h_1 < h_2 \leq D_{i,j}$  for  $i_1 \leq i \leq i_2$  and  $j_1 \leq j \leq j_2$ ,  $rec_D(i_1, i_2, k_1, k_2, h_1, h_2)$  is the 3D rectangle with the corners  $(i_1 - 1, k_1 - 1, h_1)$ ,  $(i_1 - 1, k_2, h_1)$ ,  $(i_2, k_1 - 1, h_1)$ ,  $(i_2, k_2, h_1)$ , where  $l = 1, 2$ , lying within  $his(D)$ .
3. For the matrix  $D$ ,  $r_D$  is the minimum number of 3D rectangles  $rec_D(i_1, i_2, k_1, k_2, h_1, h_2)$  which form a partition of  $his(D)$ . Note that  $r_D \leq n^2$ .

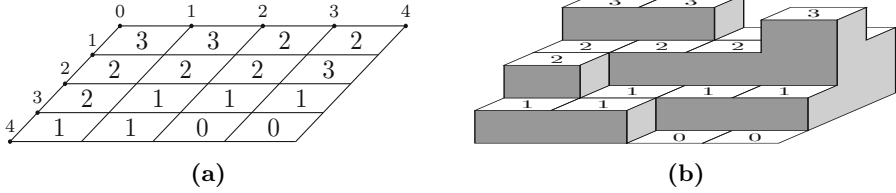


Fig. 1. (a) A matrix  $D$  on a grid, and (b) its corresponding histogram  $his(D)$

### 3.2 Algorithms

Our first geometric algorithm for nonnegative integer matrix multiplication relies on the following key lemma.

**Lemma 2.** *Let  $P_A$  be a partition of the matrix  $A$  into 3D rectangles  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$ , and let  $P_B$  be a partition of the matrix  $B$  into 3D rectangles  $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2)$ . For any  $1 \leq i \leq n, 1 \leq j \leq n$ , the entry  $C_{i,j}$  of the matrix product  $C$  of  $A$  and  $B$  is equal to the sum of  $(h_2 - h_1)(h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$ , over rectangle pairs  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A, rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$  satisfying  $i \in [i_1, i_2]$  and  $j \in [j_1, j_2]$ .*

*Proof.* For  $1 \leq l_1 < l_2 \leq n$  and  $1 \leq m_1 < m_2 \leq n$ , let  $I(l_1, l_2, m_1, m_2)$  be the  $n \times n$  0–1 matrix where  $I(l_1, l_2, m_1, m_2)_{i,k} = 1$  iff  $l_1 \leq i \leq l_2$  and  $m_1 \leq k \leq m_2$ .

Clearly, we have  $A = \sum_{rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A} (h_2 - h_1) I(i_1, i_2, k_1, k_2)$ . Similarly, we have  $B = \sum_{rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B} (h'_2 - h'_1) I(k'_1, k'_2, j_1, j_2)$ .

It follows that  $C = A \times B$  is the sum over pairs  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A, rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$  of  $(h_2 - h_1)(h'_2 - h'_1) I(i_1, i_2, k_1, k_2) \times I(k'_1, k'_2, j_1, j_2)$ . It remains to observe that  $(I(i_1, i_2, k_1 + 1, k_2) \times I(k'_1, k'_2, j_1 + 1, j_2))_{i,j} = \#[k_1, k_2] \cap [k'_1, k'_2]$  if  $i_1 < i \leq i_2$  and  $j_1 < j \leq j_2$  and it is equal to zero otherwise.  $\square$

#### Algorithm 1

*Input:* Two  $n \times n$  matrices  $A, B$  with nonnegative integer entries.

*Output:* The arithmetic matrix product  $C$  of  $A$  and  $B$ .

1. Find a partition  $P_A$  of  $his(A)$  into 3D rectangles  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$  whose number is within  $O(1)$  of the minimum.
2. Find a partition  $P_B$  of  $his(B)$  into 3D rectangles  $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2)$  whose number is within  $O(1)$  of the minimum.
3. Initialize an interval tree  $S$  on the  $k$ -coordinates of the rectangles in  $P_A$  and  $P_B$ . For each 3D rectangle  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$  insert  $[k_1, k_2]$ , with a pointer to  $A(i_1, i_2, k_1, k_2, h_1, h_2)$ , into  $S$ .
4. Initialize interval lists  $Start_j, End_j$ , for  $j = 1, \dots, n$ . For each rectangle  $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$  report all intervals  $[k_1, k_2]$  in  $S$  that intersect  $[k'_1, k'_2]$ . For each such interval  $[k_1, k_2]$ , with pointer to  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2)$ , insert the interval  $[i_1, i_2]$  with the weight  $(h_2 - h_1) \times (h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$  into the lists  $Start_{j_1}$  and  $End_{j_2}$ .

5. Initialize a weighted interval tree  $U$  on endpoints  $1, \dots, n$ . For  $j = 1, \dots, n$ , iterate the following steps. For  $j > 1$ , remove all weighted intervals  $[i_1, i_2]$  on the list  $End_{j-1}$  from  $U$ . Insert all weighted intervals  $[i_1, i_2]$  on the list  $Start_j$  into  $U$ . For  $i = 1, \dots, n$ , set  $C_{i,j}$  to the value returned by  $U$  in response to the weight query at  $i$ .

**Lemma 3.** *Let  $int(P_A, P_B)$  stand for the number of pairs  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A, rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$ , for which  $[k_1, k_2] \cap [k'_1, k'_2] \neq \emptyset$ . Algorithm 1 runs in time  $\tilde{O}(n^2 + int(P_A, P_B)) = \tilde{O}(n^2 + r_{AB})$ .*

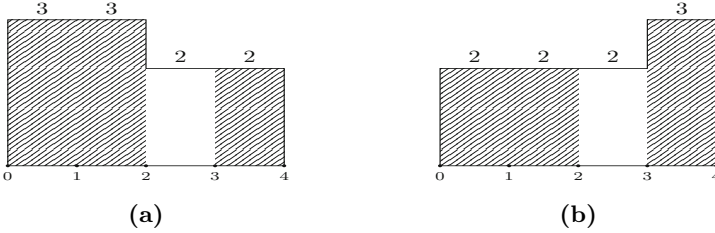
*Proof.* To implement steps 1 and 2 in  $\tilde{O}(n^2)$  time, use the algorithm from the preceding section (Theorem 1). Step 3 can be implemented in  $\tilde{O}(n + r_A + r_B) = O(n^2)$  time by Fact 1. In Step 4, the queries to  $S$  take  $\tilde{O}(int(P_A, P_B))$  time by Fact 1. In Step 5, the initialization of the data structure  $U$  takes  $\tilde{O}(n)$  time by Lemma 2. Next, the updates of the data structure  $U$  take  $\tilde{O}(int(P_A, P_B))$  time by Lemma 2, while computing all columns of  $C$  takes  $\tilde{O}(n^2)$  time by Remark 2.  $\square$

**Theorem 2.** *The matrix product of two  $n \times n$  matrices  $A, B$  with nonnegative integer entries can be computed in  $\tilde{O}(n^2 + r_A r_B)$  time.*

*Proof.* Algorithm 1 yields the theorem. Its correctness follows from Lemma 2 that basically says that for each pair of 3D rectangles,  $rec_A(i_1, i_2, k_1, k_2, h_1, h_2) \in P_A$  and  $rec_B(k'_1, k'_2, j_1, j_2, h'_1, h'_2) \in P_B$ ,  $C_{i,j}$  should be increased by  $(h_2 - h_1) \times (h'_2 - h'_1) \times \#[k_1, k_2] \cap [k'_1, k'_2]$  for  $i \in [i_1, i_2]$  and  $j \in [j_1, j_2]$ . In Step 4, two identical intervals  $[i_1, i_2]$  corresponding to the left and right edge of the submatrix of  $C$  whose entries should be increased by the aforementioned value are inserted in the lists  $Start_{j_1}$  and  $End_{j_2}$ , respectively. In both cases, they are weighted by the aforementioned value. In Step 5, in iteration  $j_1$ , the weighted interval  $[i_1, i_2]$  from  $Start_{j_1}$  is inserted into the weighted interval tree  $U$ , and in iteration  $(j_2 + 1)$ , it is removed from  $U$  as its copy is in  $End_{j_2}$ . In the iterations  $j = j_1, \dots, j_2$  in Step 5, when the interval  $[i_1, i_2]$  is kept in the weighted interval tree,  $U$  and the entries of the submatrix  $C_{i,j}$ ,  $i_1 \leq i \leq i_2, j_1 \leq j \leq j_2$ , are evaluated, the weight of the interval contributes to their value. The upper time bound follows from Lemma 3.  $\square$

When only one of the matrices  $A$  and  $B$  admits a partition of its 3D histogram into relatively few 3D rectangles and we have to assume the trivial partition of the other one into  $n^2$  3D rectangles, the upper bound of Theorem 2 in terms of  $r_A, r_B$  and  $n$  seems too weak. In this case, an upper bound in terms of  $int(P_A, P_B)$  and  $n$  in Lemma 3 may be much better. To derive a better upper bound in terms of just  $\min\{r_A, r_B\}$  and  $n$ , we shall design another algorithm based on the slicing of the 3D histogram admitting a partition into relatively few 3D rectangles.

For an  $n \times n$  matrix  $D$  with nonnegative integer entries and  $i = 1, \dots, n$ , let  $slice_i(D)$  stand for the part of  $his(D)$  between the two planes perpendicular to the  $Y$  axis whose intersection with the  $XY$  plane are the horizontal lines  $i - 1$  and  $i$  on the  $[0, n] \times [0, n]$  grid. In other words,  $slice_i(D)$  is a 3D histogram for



**Fig. 2.** Let  $slice_1(D)$  be the 2D histogram on the left and  $slice_2(D)$  the 2D histogram on the right. Differentiating strips are shaded. Here,  $gd(slice_1(D), slice_2(D)) = 2$ .

the  $i$ -th row. Also note that a  $slice_i(D)$  can be identified with a rectilinear 2D histogram; see Fig. 2 for an example. We define a *geometric distance* between two rectilinear 2D histograms  $H_1$  and  $H_2$  with a common base as the number of maximal vertical strips  $s$  such that:

1. for  $i = 1, 2$ ,  $s$  contains exactly one maximal subsegment  $e_i$  of an edge of  $H_i$  different from and parallel to the base of the histograms, and
2. the subsegments  $e_1$  and  $e_2$  do not overlap.

See Fig. 2. We shall call such strips *differentiating strips*. For  $slice_i(D)$  and  $slice_k(D)$ , we define the geometric distance  $gd(slice_i(D), slice_k(D))$  as that for the corresponding rectilinear 2D histograms.

**Lemma 4.** *For an  $n \times n$  matrix  $D$  with nonnegative integer entries,  $\sum_{i=1}^{n-1} gd(slice_i(D), slice_{i+1}(D)) = O(r_D)$  holds.*

*Proof.* Each differentiating strip contributes, possibly jointly with one or two neighboring differentiating strips, to two vertices in the projected planar graph considered in the proof of Theorem 1. Thus, it contributes to the parameter  $m'$  in the aforementioned proof with at least 1. It follows  $\sum_{i=1}^{n-1} gd(slice_i(D), slice_{i+1}(D)) \leq m'$ . Hence, the inequality  $m' \leq 4OPT$  established in the proof of Theorem 1 yields the thesis.  $\square$

**Algorithm 2**

*Input:* Two  $n \times n$  matrices  $A$  and  $B$  with nonnegative integer entries.

*Output:* The matrix product  $C$  of  $A$  and  $B$ .

1. For  $i = 1, \dots, n-1$ , find the differentiating strips for  $slice_i(A)$  and  $slice_{i+1}(A)$  and for each such strip  $s$  the indices  $k_1(s)$  and  $k_2(s)$  of the interval of entries  $A_{i,k_1(s)}, \dots, A_{i,k_2(s)}$  in the  $i$ -th row of  $A$  corresponding to it, as well as the difference  $h(s)$  between the common value of each entry in  $A_{i,k_1(s)}, \dots, A_{i,k_2(s)}$  and the common value of each entry in  $A_{i+1,k_1(s)}, \dots, A_{i+1,k_2(s)}$ .
2. For  $j = 1, \dots, n$ , iterate the following steps:
  - (a) Initialize a data structure  $T_j$  for counting partial sums of continuous fragments of the  $j$ -th column of the matrix  $B$ .
  - (b) Compute  $C_{1,j}$ .
  - (c) For  $i = 1, \dots, n-1$ , iterate the following steps:

- i. Set  $C_{i+1,j}$  to  $C_{i,j}$ .
- ii. For each differentiating strip  $s$  for  $slice_i(A)$  and  $slice_{i+1}(A)$ , compute  $\sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$  using  $T_j$  and set  $C_{i+1,j}$  to  $C_{i+1,j} + h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$ .

**Lemma 5.** *Algorithm 2 runs in  $\tilde{O}(n(n + r_A))$  time.*

*Proof.* Step 1 can be easily implemented in  $O(n^2)$  time. Step 2 (a) takes  $\tilde{O}(n)$  time according to Fact 2 while Step 2 (b) can be trivially implemented in  $O(n)$  time. Finally, based on Step 1, Step 2 (c) (ii) takes  $\tilde{O}(gd(slice_i(D), slice_{i+1}(D)))$  time. It follows that Step 2 (c) can be implemented in  $\tilde{O}(\sum_{i=1}^{n-1} gd(slice_i(A), slice_{i+1}(A)))$  time, i.e., in  $\tilde{O}(r_A)$  time by Lemma 4. Consequently, Step 2 takes  $\tilde{O}(n(n + r_A))$  time.  $\square$

**Theorem 3.** *The arithmetic matrix product of two  $n \times n$  matrices  $A, B$  with nonnegative integer entries can be computed in  $\tilde{O}(n(n + \min\{r_A, r_B\}))$  time.*

*Proof.* The correctness of Algorithm 2 follows from the observation that a differentiating strip  $s$  for  $slice_i(A)$  and  $slice_{i+1}(A)$  yields the difference  $h(s) \sum_{k=k_1(s)}^{k_2(s)} B_{k,j}$  between  $C_{i+1,j}$  and  $C_{i,j}$  just on the fragment corresponding to  $A_{i,k_1(s)}, \dots, A_{i,k_2(s)}$  and  $A_{i+1,k_1(s)}, \dots, A_{i+1,k_2(s)}$ , respectively. Lemma 5 yields the upper bound  $\tilde{O}(n(n + r_A))$ . The symmetric one  $\tilde{O}(n(n + r_B))$  follows from the equalities  $AB = (B^T A^T)^T$ ,  $his(B) \equiv his(B^T)$ , and consequently  $r_B = r_{B^T}$ .  $\square$

In Algorithm 2, the linear order in which the  $C_{i,j}$  are updated to  $C_{i+1,j}$  for  $i = 1, \dots, n - 1$ , along the row order of the matrix  $A$  is not necessarily optimal. Following the Boolean case [3, 5], it may be more efficient to update  $C_{i,j}$  while traversing a minimum spanning tree for the slices of  $his(A)$  under the geometric distance. Here, however, we encounter the difficulty of constructing such an optimal spanning tree or a close approximation in substantially subcubic time. The next lemma will be useful.

**Lemma 6.** *Consider the family of rectilinear planar histograms with the base  $[0, n]$ ,  $n \geq 2$  and integer coordinates of its vertices in  $[0, 2^M - 2]$ ,  $M = O(\log n)$ . There is a simple  $O(n)$ -time transformation of any histogram  $H$  in the family into an  $0 - 1$  string  $t(H)$ , such that for any  $H_1$  and  $H_2$  in the family  $gd(H_1, H_2) \leq ch(t(H_1), t(H_2)) \leq Mgd(H_1, H_2)$ , where  $ch(\cdot, \cdot)$  stands for the Hamming distance.*

*Proof.* Any histogram  $H$  in the family is uniquely represented by the vector  $(H[1], \dots, H[n]) \in \{1, \dots, 2^M - 1\}^n$ , where  $H[1], \dots, H[n]$  are the values of  $Y$  coordinates of the points on the “roof” of  $H$  increased by one with  $X$  coordinates  $0.5, 1.5, \dots, n - 0.5$  respectively.

For any  $y \in \{0, \dots, 2^M - 1\}$  denote its binary representation of length exactly  $M$  (padded with leading zeros if necessary) as  $\text{bin}(y)$ .

$$\text{Let } f(H, i) = \begin{cases} \text{bin}(H[i]), & i = 1 \vee i > 1 \wedge H[i] \neq H[i - 1] \\ \text{bin}(0), & \text{otherwise.} \end{cases}$$

The transformation  $t$  is then defined as  $t(H) = f(H, 1) \dots f(H, n)$ . We have  $ch(t(H_1), t(H_2)) = \sum_{i=1}^n ch(f(H_1, i), f(H_2, i))$  and

$$gd(H_1, H_2) = \begin{cases} 1, & H_1[1] \neq H_2[1] \\ 0, & \text{otherwise} \end{cases} + \\ + \sum_{i=2}^n \begin{cases} 1, & (H_1[i] \neq H_1[i-1] \vee H_2[i] \neq H_2[i-1]) \wedge (H_1[i] \neq H_2[i]) \\ 0, & \text{otherwise.} \end{cases}$$

Consider all possibilities that contribute exactly one to  $gd(H_1, H_2)$ :

1.  $H_1[1] \neq H_2[1]$ . In this case  $f(H_1, 1) = \text{bin}(H_1[1])$ ,  $f(H_2, 1) = \text{bin}(H_2[1])$  and  $0 \leq ch(\text{bin}(H_1[1]), \text{bin}(H_2[1])) \leq M$ .
2.  $2 \leq i \leq n \wedge H_1[i] \neq H_1[i-1] \wedge H_2[i] = H_2[i-1] \wedge H_1[i] \neq H_2[i]$ . In this case  $f(H_1, i) = \text{bin}(H_1[i])$ ,  $f(H_2, i) = \text{bin}(0)$  and  $1 \leq ch(\text{bin}(H_1[i]), \text{bin}(0)) \leq M$ .
3.  $2 \leq i \leq n \wedge H_1[i] = H_1[i-1] \wedge H_2[i] \neq H_2[i-1] \wedge H_1[i] \neq H_2[i]$ . See case 2.
4.  $2 \leq i \leq n \wedge H_1[i] \neq H_1[i-1] \wedge H_2[i] \neq H_2[i-1] \wedge H_1[i] \neq H_2[i]$ . See case 1.

To complete the proof, observe that in all other cases  $ch(f(H_1, i), f(H_2, i)) = 0$ .  $\square$

**Fact 3** [6]. For  $\epsilon > 0$ , a  $(1 + \epsilon)$ -approximation minimum spanning tree for a set of  $n$  points in  $R^d$  with integer coordinates in  $O(1)$  under the  $L_1$  or  $L_2$  metric can be computed by a Monte Carlo algorithm in  $O(dn^{1+1/(1+\epsilon)})$  time.

By combining the transformation of Lemma 6 with Fact 3 applied to the  $L_1$  metric in  $\{0, 1\}^n$  and selecting  $\epsilon = \log n$ , we obtain a Monte Carlo  $O(\log^2 n)$ -approximation algorithm for the minimum spanning tree of the slices of  $his(A)$  under the geometric distance, which runs in  $\tilde{O}(n^2)$  time. This yields a generalization of Algorithm 2 to Algorithm 3, described in the full version of our paper. By an analysis of Algorithm 3 analogous to that of Algorithm 2 and a proof analogous to that of Theorem 3, we obtain a randomized generalization of Theorem 3:

**Theorem 4.** Let  $A, B$  be two  $n \times n$  matrices  $A, B$  with nonnegative integer entries in  $[0, n^{O(1)}]$ . Next, for  $D \in \{A, B^T\}$ , let  $M_D$  be the minimum cost of a spanning tree of  $\text{slice}_i(D)$  for  $i = 1, \dots, n$ . The arithmetic matrix product of  $A$  and  $B$  can be computed by a randomized algorithm in  $\tilde{O}(n(n + \min\{M_A, M_{B^T}\}))$  time with high probability.

## 4 Final Remarks

A natural question is: Would it help to apply an algorithm that optimally rectangularizes the 2D projection in Section 2.2? Although it would yield improved results in certain cases, it would not give a better approximation factor than 4 in general for the minimum rectangular 3D partition. An example of this is when the optimal 3D partition consists of  $k$  cubes lying on top of each other.

Then the 2D projection is  $k$  concentric squares of different sizes and an optimal rectangulation of the corresponding 2D projection consists of  $4k - 3$  rectangles. Hence, for large  $k$ , the approximation factor tends to 4.

The 4-approximation algorithm for minimum rectangular partition of a 3D histogram in case the histogram is  $his(D)$  for an input  $n \times n$  matrix  $D$  with nonnegative integer entries can easily be implemented in  $O(n^2)$  time. Also note that the resulting partition of  $his(D)$  can be used to form a compressed representation of  $D$  requiring solely  $\tilde{O}(r_D)$  bits if the values of the entries in  $D$  are  $n^{O(1)}$ -bounded.

Our geometric algorithms for integer matrix multiplication can also be applied to derive faster  $(1+\epsilon)$ -approximation algorithms for integer matrix multiplication; if the range of an input matrix  $D$  is  $[0, n^{O(1)}]$ , then round each entry to the smallest integer power of  $(1+\epsilon)$  that is not less than the entry. The resulting matrix  $D'$  has only a logarithmic number of different entry values and hence  $r_{D'}$  may be much less than  $r_D$ .

Our algorithms and upper time bounds for integer  $n \times n$  matrix multiplication can easily be extended to include integer rectangular matrix multiplication.

## References

1. Bansal, N., Williams, R.: Regularity Lemmas and Combinatorial Algorithms. *Theory of Computing* **8**(1), 69–94 (2012)
2. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: *Computational Geometry: Algorithms and Applications*. 3rd edn. Springer, Santa Clara (2008)
3. Björklund, A., Lingas, A.: Fast boolean matrix multiplication for highly clustered data. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 2001. LNCS, vol. 2125, p. 258. Springer, Heidelberg (2001)
4. Dielissen, V.J., Kaldewai, A.: Rectangular Partition is Polynomial in Two Dimensions but NP-Complete in Three. *Information Processing Letters* **38**(1), 1–6 (1991)
5. Gasieniec, L., Lingas, A.: An Improved Bound on Boolean Matrix Multiplication for Highly Clustered Data. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 329–339. Springer, Heidelberg (2003)
6. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: *Proc. of STOC 1998*, pp. 604–613 (1998)
7. Keil, J.M.: *Polygon Decomposition*. Survey, Dept. Comput. Sc. Univ. Saskatchewan (1996)
8. Le Gall, F.: Powers of Tensors and Fast Matrix Multiplication. In: *Proc. of the 39th ISSAC*, pp. 296–303 (2014)
9. Lingas, A.: A Geometric Approach to Boolean Matrix Multiplication. In: Bose, P., Morin, P. (eds.) *ISAAC 2002*. LNCS, vol. 2518, pp. 501–510. Springer, Heidelberg (2002)
10. Lipski, W.: Finding a Manhattan path and related problems. *Networks* **13**(3), 399–409 (1983)

11. Mehlhorn, K.: Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry. EATCS Monographs on Theo. Comput. Sc., Springer (1984)
12. Muthukrishnan, S., Poosala, V., Suel, T.: On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 236–256. Springer, Heidelberg (1998)
13. Sack, J.-R., Urrutia, J. (ed): Handbook of Computational Geometry. Elsevier (2000)



# **Graph Algorithms: Enumeration**

# Enumeration of Maximum Common Subtree Isomorphisms with Polynomial-Delay

Andre Droschinsky<sup>1</sup>(✉), Bernhard Heinemann<sup>1</sup>, Nils Kriege<sup>2</sup>,  
and Petra Mutzel<sup>2</sup>

<sup>1</sup> Faculty of Mathematics and Computer Science,  
FernUniversität in Hagen, Hagen, Germany  
{andre.droschinsky,bernhard.heinemann}@fernuni-hagen.de

<sup>2</sup> Department of Computer Science, Technische  
Universität Dortmund, Dortmund, Germany  
{nils.kriege,petra.mutzel}@tu-dortmund.de

**Abstract.** The maximum common subgraph problem asks for the maximum size of a common subgraph of two given graphs. The problem is NP-hard, but can be solved in polynomial time if both, the input graphs and the common subgraph are restricted to trees. Since the optimal solution of the maximum common subtree problem is not unique, the problem of enumerating all solutions, i.e., the isomorphisms between the two subtrees, is of interest. We present the first polynomial-delay algorithm for the problem of enumerating all maximum common subtree isomorphisms between a given pair of trees. Our approach is based on the algorithm of Edmonds for solving the maximum common subtree problem using a dynamic programming approach in combination with bipartite matching problems. As a side result, we obtain a polynomial-delay algorithm for enumerating all maximum weight matchings in a complete bipartite graph. We show how to extend the new approach in order to enumerate all solutions of the maximum weighted common subtree problem and to the maximal common subtree problem. Our experimental evaluation on both, randomly generated as well as real-world instances, demonstrates the practical usefulness of our algorithm.

## 1 Introduction

In many application areas such as pattern recognition [4], or chem- and bioinformatics [10],[16], it is an important task to elucidate similarities between structured objects like proteins or small molecules. A widely-used and successful approach regarding this is to model objects as graphs and to identify their maximum common subgraphs (MCSs). As a MCS apparently is not unique, it is of interest to find *all* solutions. Since the number of solutions may be superexponential in the input size, the running time cannot be expected to be polynomial in this case. For this reason, enumeration algorithms are said to have *polynomial*

---

This work was supported by the German Research Foundation (DFG), priority programme “Algorithms for Big Data” (SPP 1736).

*total time* if the running time is bounded by a polynomial in the input size and the number of solutions. They have *polynomial-delay* if the running time before the output of the first solution, and after the output of a solution until providing the next solution or halting, is polynomially bounded in the input size [8].

Unfortunately, MCS is known to be NP-hard and consequently a polynomial-time algorithm is not even known for finding a single maximum solution. However, it is straightforward to determine an isomorphism between common subgraphs that is maximal with respect to inclusion, but not necessarily maximum possible. The enumeration of maximal common subgraph isomorphisms has been considered in several papers, and algorithms are typically based on the one-to-one-correspondence of isomorphisms between common subgraphs and cliques in a product graph [13]. As a matter of fact it is possible to enumerate maximal common subgraph isomorphisms with polynomial-delay, since maximal cliques can be listed with polynomial-delay [8]. In practice a common approach is to utilize the Bron-Kerbosch algorithm [2] for listing maximal cliques, which, however, does not yield a polynomial total time algorithm. Koch [3],[10] modified this algorithm to enumerate cliques corresponding to maximal isomorphisms between connected common subgraphs. Although not providing any guarantee in terms of running time, this approach is considered to be more practical since the number of solutions is drastically reduced.

Only few tractable variants of MCS are known. Edmonds was reported [14] to have proposed a polynomial time algorithm for solving the maximum common subtree problem, where the input graphs and the desired common subgraph are trees, by means of maximum weight bipartite matching. Related problems like the *maximum agreement subtree* of rooted trees are well-studied and are, for example, considered in [9] using similar ideas to those of Edmonds [14]. Polynomial time algorithms were presented to find connected MCSs in outerplanar graphs under the additional requirement that blocks, i.e., maximal biconnected subgraphs, and bridges of the input graphs are preserved [17]. The unrestricted problem can be solved in polynomial time in outerplanar graphs of bounded degree [1]. The problem also is tractable if one of the input graphs is a bounded-degree partial  $k$ -tree and the other is a connected graph with a polynomial number of spanning trees [19]. Recently, a polynomial-time algorithm to find a biconnected MCS in series-parallel graphs has been developed [11]. However, no enumeration algorithms with polynomial total time or polynomial-delay have been proposed for any of these efficiently solvable variants of MCS by now.

**Our Contribution.** We address the problem to enumerate all isomorphisms between maximum common subtrees of two given trees and present a polynomial-delay algorithm. We utilize the idea of Edmonds' algorithm to decompose trees into subtrees and solve subproblems by weighted bipartite matching. In order to enumerate maximum weight bipartite matchings with polynomial-delay we propose a new technique that is based on Uno's algorithm for listing perfect matchings [18]. The isomorphisms subject to enumeration correspond to combined solutions of multiple matching problems. Interrupting the enumeration process of matchings for one instance in order to proceed with a different matching

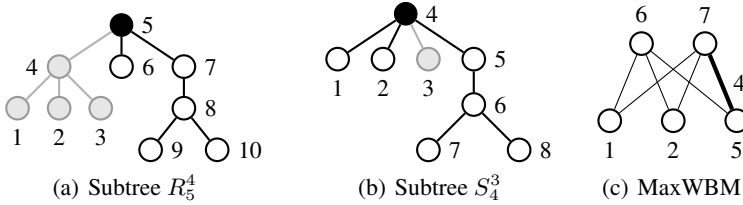
problem, allows us to combine partial solutions and to output all maximum isomorphisms without duplicates with polynomial-delay using only polynomial space. Going beyond that, we present modifications to enumerate maximum weight and maximal solutions. In an experimental evaluation we show that our algorithms are efficient in practice on synthetic and real-world data sets from cheminformatics. Our method is shown to outperform the algorithm proposed by Koch [10] for arbitrary graphs on tree instances. The techniques we propose can be taken as a basis for finding efficient enumeration algorithms for MCS problems in more complex graph classes.

## 2 Preliminaries

In this paper,  $G = (V, E)$  is a *simple undirected graph*. We call  $v \in V$  a *vertex* and  $e = uv = vu \in E$  an *edge* of  $G$ . For a graph  $G = (V, E)$  we define  $V(G) := V$  and  $E(G) := E$ . Two graphs  $G_1 = (V_1, E_1)$ ,  $G_2 = (V_2, E_2)$  are *isomorphic* if there is a bijective function (an isomorphism)  $\varphi : V_1 \rightarrow V_2$  such that  $uv \in E_1 \Leftrightarrow \varphi(u)\varphi(v) \in E_2$ . The *size*  $|\varphi|$  of an isomorphism is defined as the number of vertices in each of the sets  $V_1, V_2$ , i.e.,  $|\varphi| := |V_1| = |V_2|$ . A *forest* is a graph without cycles; a *tree* is a connected forest. A tree  $T = (V, E)$  with an explicit *root* vertex  $r \in V$  is a *rooted tree*. Let  $R$  and  $S$  be trees. If subtrees  $R'$  of  $R$  and  $S'$  of  $S$  are isomorphic, we call an isomorphism  $\varphi : V(R') \rightarrow V(S')$  a *common subtree isomorphism (CSTI)* of  $R$  and  $S$ . If the input trees are clear from the context, we omit them. A CSTI  $\varphi$  is a *maximum common subtree isomorphism (maximum CSTI)* if there are no other CSTI of  $R$  and  $S$  with size greater than  $|\varphi|$ . A CSTI  $\varphi$  is *maximal* if there is no CSTI  $\varphi'$  with  $|\varphi'| > |\varphi|$  and  $\varphi(x) = \varphi'(x)$  for all  $x \in R' = \text{dom}(\varphi)$ . For a graph  $G = (V, E)$  a matching  $M \subseteq E$  is a set of edges, such that no two edges share the same vertex. A matching  $M$  of  $G$  is *maximal* if there is no other matching  $M' \supsetneq M$  of  $G$ . It is *perfect*, if  $2 \cdot |M| = |V|$ . A *weighted graph* is a graph together with a function  $w : E \rightarrow \mathbb{Q}$ . For a matching  $M$  of a weighted graph we define its weight by  $W(M) := \sum_{e \in M} w(e)$ . If the vertices of a graph  $G$  can be separated into exactly two disjoint sets  $V, X$  such that  $E(G) \subseteq V \times X$ , then the graph is called *bipartite*. In many cases the disjoint sets are already given as part of the input. In this case we write  $G = (V \cup X, E)$ , where  $E \subseteq V \times X$ . We call a matching  $M$  of a weighted bipartite graph  $G$  a *maximum weight bipartite matching (MaxWBM)* if there is no other matching  $M'$  of  $G$  with  $W(M') > W(M)$ . We denote a perfect MaxWBM by *MaxWBPM*.

### 2.1 Edmonds' Algorithm

Our approach is related to Edmonds' algorithm, which solves the maximum common subtree problem by means of maximum weight bipartite matching. Consider a tree  $T$  and an edge  $e = uv \in E(T)$  of the tree. By removing  $e$  we obtain two subtrees. We denote by  $T_u^v$  the rooted subtree not containing  $v$  with root vertex  $u$  and refer to the other rooted subtree by  $\overline{T_u^v} := T_v^u$ , cf.



**Fig. 1.** Two rooted subtrees (a) and (b) and the associated weighted bipartite matching problem (c). Gray vertices and edges are not part of the subtrees, root vertices are shown in solid black; edges without label in (c) have weight 1.

Fig. 1. For every pair of rooted subtrees of the two input trees  $R$  and  $S$ , the size of a maximum CSTI is computed under the restriction that the isomorphism maps the two roots to each other. The result is stored in a table  $D$  by dynamic programming. Let  $R_s^i$  and  $S_t^j$  be two rooted subtrees and  $V = \{s_1, \dots, s_n\}$  and  $X = \{t_1, \dots, t_m\}$  the children of  $s$  in  $R_s^i$  and  $t$  in  $S_t^j$ , respectively. Then  $D(R_s^i, S_t^j) = 1 + M$ , where  $M$  is the size of a MaxWBM in the complete bipartite graph with the vertex set  $V \cup X$ . The edge weights  $w$  of this graph are determined by the entries in  $D$  for pairs of smaller rooted subtrees according to  $w(s_k, t_l) = D(R_{s_k}^s, S_{t_l}^t)$ , where  $k \in \{1, \dots, n\}$ ,  $l \in \{1, \dots, m\}$ . The matching defines the mapping of the children of the two roots, cf. Fig. 1. The table  $D$  is filled by ordering the subtrees according to increased size of subtrees, thus the required partial solutions are always available from  $D$ . Finally, the maximum size of a CSTI is determined by combining all pairs of corresponding rooted subtrees. That is, the result for  $R_s^i$  and  $R_i^s$  combined with  $S_t^j$  and  $S_j^t$  is  $D(R_s^i, S_t^j) + D(R_i^s, S_j^t)$ , for any edge  $is \in E(R), jt \in E(S)$ .

### 3 Enumeration of Maximum Weight Matchings with Polynomial-Delay

In this section we develop a polynomial-delay algorithm to enumerate all maximum weight matchings in a complete bipartite graph. In Edmonds’ algorithm, cf. Sect. 2.1, the size of a maximum common subtree isomorphism of two trees  $R$  and  $S$  is calculated by determining MaxWBMs and using their weights for the calculations. An approach to enumerate all maximum CSTIs of  $R$  and  $S$  is to enumerate all the MaxWBMs in these graphs. We do this by reducing the enumeration of MaxWBMs in a graph  $G'$  to the enumeration of perfect matchings in another graph  $G^*$ , as described below.

The bipartite weighted graphs  $G' = (V' \cup X', E')$  that occur in Edmonds’ algorithm, cf. Fig. 1, are complete, i.e.,  $E' = V' \times X'$ . All the edge weights in these graphs are positive. Due to this fact, every maximum weight bipartite matching is maximal. We use this in our algorithm to achieve a reduction from MaxWBM to MaxWBPM. The reduction from the weighted problem to an unweighted



**Fig. 2.** Different matchings (thick edges) in the extended graph, which correspond to the same matching in the initial graph (black vertices and edges only).

problem is achieved with help of the well-known Hungarian method [7], [12]. Finally we use a modification of an enumeration algorithm for perfect matchings in a bipartite graph [18] to obtain our MaxWBMs. We describe the steps in detail.

First, we show the reduction from the enumeration of MaxWBMs in  $G'$  to the enumeration of MaxWBPMs in another graph  $G$ . A perfect matching in a bipartite graph requires the same number of vertices in both vertex sets. Therefore we extend  $G'$  by additional vertices and edges if  $|V'| \neq |X'|$ . Assume w.l.o.g.  $|V'| < |X'|$ , then we add vertices  $U$  and edges with zero weight between  $U$  and  $X'$  such that  $|V'| + |U| = |X'|$ . We refer to the resulting graph  $G := (V \cup X, V \times X) := ((V' \cup U) \cup X', (V' \cup U) \times X')$  as the *extended graph*. It is obvious that the weight of a MaxWBPM of  $G$  is identical to the weight of a MaxWBM of  $G'$ . The construction of  $G$  implies that for every MaxWBM  $M'$  of  $G'$  there is at least one MaxWBPM  $M \supseteq M'$  of  $G$ . The issue that there may be more than one, will be addressed later.

The problem of finding all *minimum* weight matchings in  $G$  can be reduced to finding all perfect matchings in the *admissible subgraph*  $G^*$  of an optimal dual solution based on the standard LP formulation for the minimum weighted matching problem of  $G$ . This is implied by the well-known *Complementary Slackness Theorem*. The admissible subgraph and its construction is described in [5], [7]. Note that  $G^*$  and  $G$  share the same vertices,  $E(G^*) \subseteq E(G)$ , and  $G^*$  is unweighted. The reduction from maximum weight matchings is very similar. For example the weights of  $G$  can be multiplied by  $-1$  before calculating an optimal dual solution and the admissible subgraph  $G^*$ .

We obtain all maximum weight bipartite matchings of the extended graph  $G$  by enumerating all the perfect matchings in the admissible subgraph  $G^*$ . Removing the additional vertices and edges from  $G$  yields all maximum weight matchings of  $G'$ . Unfortunately, two different matchings  $M_1, M_2$  of  $G$  can lead to the same matching  $M$  of  $G'$ , as is shown in Fig. 2. We handle this by directly modifying Uno's algorithm for the enumeration of all perfect matchings in a bipartite graph [18].

First, we briefly describe Uno's algorithm. Given a bipartite graph and a (first) perfect matching  $M$ , one edge  $e \in M$  is selected. The problem is then divided into the enumeration of the perfect matchings containing  $e$  and those not containing  $e$ . These subproblems lead to a graph  $G^+(e)$  with initial matching  $M$  and another graph  $G^-(e)$  with initial matching  $M'$  as described below. Both

graphs have less vertices and/or edges than the previous graph. The enumeration continues recursively until no more edge  $e$  can be selected. The selection of  $e$  is key to the algorithm. Uno proves that there is another perfect matching iff there is an alternating cycle, i.e., a cycle of even length, where exactly every second edge is part of the matching. If  $e$  is part of an alternating cycle, another perfect matching  $M'$  can be obtained by exchanging the edges along the cycle, i.e., an edge, that was part of the matching  $M$ , will no longer be included in the matching, and vice versa. Edges that are not part of any cycle are removed in each recursive step of the algorithm.

On a bipartite graph with  $n$  vertices and  $m$  edges, a perfect matching can be calculated in time  $\mathcal{O}(n^{1/2}m)$  [18]. This is the first step in Uno's algorithm. In our case the initial graph is the bipartite admissible subgraph  $G^*$ ; the perfect matching is the one obtained during the calculation of  $G^*$ . Uno states  $\mathcal{O}(n+m)$  time per additional matching, which basically is the time to find an alternating cycle, and improves this with amortized cost analysis to  $\mathcal{O}(n)$ .

If the search for an alternating cycle starts from a vertex that is part of  $V'$  (black vertices in top row in Fig. 2), the newly obtained matching  $M'$  will be different from the previous matching  $M$  regarding  $G'$ , i.e.,  $M' \cap E(G') \neq M \cap E(G')$ . If there is no such vertex, the current recursion has finished, as no new matchings regarding  $G'$  can be found. In this sense we prune the recursion tree. Unfortunately, this means that the time per matching,  $\mathcal{O}(n)$ , is not valid for our modification, since our algorithm stops the recursion as soon as there is no more vertex  $v \in V'$ . I.e., the higher costs of the first matchings cannot be divided to the costs of the later matchings. As an example, in a nearly complete graph with only one matching regarding  $G'$ , time  $\mathcal{O}(n^2)$  is needed to prove there is no alternating cycle containing a vertex of  $V'$ .

We now take a closer look at the recursion tree. Whenever an alternating cycle is found, we obtain a new matching and the two subproblems  $G^+(e)$  and  $G^-(e)$  mentioned above. If we do not find a cycle, the current recursion halts. Hence, every vertex in the recursion tree has exactly 0 or 2 children. Therefore, a recursion tree with  $n$  nodes yields  $(n-1)/2$  matchings and we obtain our first lemma.

**Lemma 1.** *All  $N$  maximum weight matchings of any bipartite graph  $G' = (V' \cup X', V' \times X')$  can be enumerated with polynomial-delay in total time  $\mathcal{O}(n^3 + Nn^2)$ , where  $n = |V'| + |X'|$*

*Proof.* The admissible subgraph  $G^*$  of an optimal dual solution and the first perfect matching are obtained in time  $\mathcal{O}(n^3)$  according to the Hungarian method. They are the starting point for the modification of Uno's algorithm. The total time to enumerate all other perfect matchings is bounded by  $\mathcal{O}(Nn^2)$ , as stated above. Note that  $|E(G^*)| \in \mathcal{O}(n^2)$ . The recursion in Uno's algorithm stops as soon as there is no more alternating cycle containing a vertex of  $V'$ . At any time there can be no more than  $|E(G^*)|$  recursions on the stack. Therefore our algorithm to enumerate all MaxWBMs of  $G'$  is a polynomial-delay algorithm with total time  $\mathcal{O}(n^3 + Nn^2)$ .  $\square$

Note that the calculation of the admissible subgraph is needed only once for every pair of rooted subtrees. If the two disjoint sets of  $V(G')$  have the same size, we achieve a total time of  $\mathcal{O}(n^3 + Nn)$ , because the recursion in our algorithm is identical to the recursion in Uno's algorithm.

## 4 Enumeration of Maximum Common Subtree Isomorphisms with Polynomial-Delay

In this section we present a polynomial-delay algorithm for the enumeration of all maximum common subtree isomorphisms of given trees  $R$  and  $S$ . We assume that both trees have at least two vertices. Otherwise, the enumeration of the maximum CSTIs is trivial. Our algorithm is based on Edmonds' algorithm (Sect. 2.1). First, we describe the idea of acquiring a single maximum CSTI of  $R$  and  $S$ . Then we outline our enumeration algorithm, before we go into detail.

To obtain a maximum CSTI  $\varphi$  of  $R$  and  $S$ , we first determine the size  $m$  of a maximum CSTI with Edmonds' algorithm. Then we consider pairs of rooted subtrees  $(r, s) := (R_v^u, S_x^w)$  with  $D(r, s) + D(\bar{r}, \bar{s}) = m$ . As described in Sec. 2.1, we map  $\varphi(u) = w$ ,  $\varphi(v) = x$ . A MaxWBM of the bipartite graph of the children of  $v$  and  $x$  determines the mapping of these children (see Sec. 2.1). The same applies to the children of the vertices  $u, w$ .

The idea to find all the maximum CSTIs of  $R$  and  $S$  is to enumerate all MaxWBMs (instead of calculating only one) per bipartite graph and to combine all possible solutions. To do this, we enumerate all pairs  $(r, s)$  of rooted subtrees where  $D(r, s) + D(\bar{r}, \bar{s}) = m$ . For each of these pairs we enumerate all maximum CSTIs and combine them with all maximum CSTIs of  $(\bar{r}, \bar{s})$ .

By enumerating a non-maximum CSTI on  $(r, s)$  or  $(\bar{r}, \bar{s})$ , the total size of the CSTI of  $R$  and  $S$  would be less than  $m$ , thus no maximum. And by enumerating and combining all maximum CSTIs of  $(r, s)$  and  $(\bar{r}, \bar{s})$ , we do not miss any maximum CSTIs of  $R$  and  $S$ . For every maximum CSTI  $\varphi$  of  $R$  and  $S$ , the edges of the matchings are directly given by the isomorphism; if  $\varphi(a) = b$ , then the edge  $ab$  is included in a matching of the corresponding bipartite graph. All the matchings have to be MaxWBMs, otherwise  $\varphi$  would be no maximum CSTI. Thus this approach is correct.

First, we iterate through all rooted subtrees  $r = R_v^u$  of  $R$ . For each subtree  $r$  we iterate through all rooted subtrees  $s = S_x^w$  of  $S$  with  $D(r, s) + D(\bar{r}, \bar{s}) = m$ . For each of these pairs  $(r, s)$  we enumerate all MaxWBMs of the bipartite graph of the children of  $v, x$  with our algorithm described in Sec. 3. The matchings determine the mappings of the children, thus adding vertices to the currently enumerated CSTI. Let  $M = \{v_1x_1, \dots, v_lx_l\}$  be a MaxWBM. Then we recursively enumerate on the pairs  $(R_{v_k}^v, S_{x_k}^x)$  of rooted subtrees,  $\forall k \in \{1, \dots, l\}$ , where  $D(R_{v_k}^v, S_{x_k}^x) > 1$ , i.e., both subtrees are no single vertices. This means, we first determine the first maximum CSTI on all of these pairs of rooted subtrees. All these CSTIs together with the previous mappings ( $\varphi(v) = x, \varphi(v_1) = x_1, \dots$ ) give us the first maximum CSTI of the pair  $(r, s)$  of rooted subtrees. We acquire the second maximum CSTI of  $(r, s)$  by enumerating the second CSTI on the



pair  $(R_{v_l}^v, S_{x_l}^x)$  and so on, until there are no more. Then we calculate the second CSTI on  $(R_{v_{l-1}}^v, S_{x_{l-1}}^x)$  and enumerate all maximum CSTIs on  $(R_{v_l}^v, S_{x_l}^x)$  again. We repeat this until we enumerated all combinations, i.e., the last CSTIs of all pairs  $(R_{v_k}^v, S_{x_k}^x)$ . For every calculated maximum CSTI of  $(r, s)$  we enumerate, as stated above, all maximum CSTIs of  $(\bar{r}, \bar{s})$  and combine them to a maximum CSTI of  $R$  and  $S$  and output the isomorphisms.

Although we do not miss any maximum CSTI with the presented algorithm, and our algorithm from Sec. 3 enumerates every MaxWBM of the corresponding weighted bipartite graph exactly once, the algorithm for maximum CSTIs outputs every maximum CSTI  $\varphi$  twice or more. Consider the trees  $R = (\{a, b, c\}, \{ab, bc\})$  and  $S = (\{1, 2, 3\}, \{12, 23\})$ . By selecting  $r = R_a^a, s = S_2^2$  we directly obtain  $\varphi(a) = 1, \varphi(b) = 2$ . The bipartite graph  $G' = (\{c, 3\}, \{c3\})$  with weight  $w(c3) = 1$  yields  $\{c3\}$  as a MaxWBM and therefore  $\varphi(c) = 3$ . If we select  $r = R_a^b, s = S_1^2$ , or  $r = R_c^b, s = S_3^2$ , or  $r = R_b^c, s = S_2^3$ , we obtain the same maximum CSTI  $\varphi$ . In general every maximum CSTI is reported twice for every edge of the common subtree.

The problem can be solved by adding another step to our algorithm. Assume  $r = R_v^u$  and the enumeration on this rooted subtree is finished. Before selecting the next rooted subtree of  $R$ , we delete the edge  $uv$  from  $R$  and recalculate the table  $D$ . Edmonds' algorithm (the calculation of  $D$ ) works on forests as well as on trees, so the missing edge is no problem here. We call the new graph  $R'$ . If the size of a maximum CSTI of  $(R', S)$  is less than the size of a maximum CSTI of  $(R, S)$  our algorithm halts. With the selection of  $R_v^u$  we enumerate every and only maximum CSTI which contain both vertices  $u$  and  $v$ . After deleting the edge  $uv$ , all subsequently enumerated maximum CSTI will contain at most one of the two vertices  $u, v$ , as they are in different connected components in  $R'$ . This means, all maximum CSTIs which were enumerated before removing  $uv$  differ from all maximum CSTIs after removing  $uv$ . With the last modification we have an algorithm that enumerates every maximum CSTI of trees  $R$  and  $S$  exactly once.

We now analyze the time and space complexity of our algorithm. Every enumerated maximum CSTI  $\varphi$  is determined by the rooted subtrees  $(r, s)$  and maximum weight bipartite matchings  $M_i$  in some bipartite graphs  $G'_i, 1 \leq i \leq k$ , for some  $k < |\varphi|$ . Let  $r_i, s_i$  be the number of vertices of trees  $R, S$  in  $G'_i$  and  $m_i := \max\{r_i, s_i\}$ . Because the vertices in all the bipartite weighted graphs  $G'_i$  are pairwise disjoint, we get  $\sum_i r_i < |V(R)|$  and  $\sum_i s_i < |V(S)|$ . Given the initial optimal solutions for the matching problems that are computed by Edmonds' algorithm, the amortized time to calculate a matching  $M_i$  is bounded by  $\mathcal{O}(m_i^2)$ , see Lemma 1. We have  $\sum_i m_i^2 = \sum_i \max\{r_i, s_i\}^2 = \sum_{\{i|r_i > s_i\}} r_i^2 + \sum_{\{i|r_i \leq s_i\}} s_i^2 < |V(R)|^2 + |V(S)|^2$ . Therefore the amortized time to enumerate a single maximum CSTI is bounded by  $O(|V(R)|^2 + |V(S)|^2)$  excluding the calculation of the table  $D$ . A worst case example is a pair of trees  $R, S$ , where  $R$  is a path, i.e., all vertices have degree 1 or 2, and  $S$  contains exactly one path of  $|V(R)|$  vertices.

The table  $D$  is calculated every time a new rooted subtree  $r$  is selected. After  $|v(R)| + 2 - |\varphi|$  calculations  $|v(R)| + 1 - |\varphi|$  edges have been removed from  $R$ , so there are only  $|\varphi| - 1$  vertices left, which is less than the size  $|\varphi|$  of a maximum CSTI. This means, there is no other maximum CSTI of  $(R, S)$  and our algorithm has stopped. Each calculation of  $D$  requires the calculation of  $\mathcal{O}(|E(R)| \cdot |E(S)|)$  optimal dual solutions and admissible subgraphs. The time for each calculation is bounded by  $\mathcal{O}(\max\{|E(R)|, |E(S)|\}^3)$ . This worst case running time occurs for trees  $R, S$ , which are both "stars", i.e., all vertices of both trees are connected to a single center vertex.

The total time for all calculations of  $D$  is polynomial, the time for each maximum CSTI is also bounded by a polynomial, as each MaxWBM is calculated in polynomial time. This proves that we have a polynomial-delay algorithm. The total space needed for our algorithm is basically determined by the admissible subgraphs. Each of them can be stored in space  $\mathcal{O}(|V(R)| + |V(S)|)$  [5]. Thus, we have achieved our goal.

**Theorem 1.** *It is possible to enumerate all  $N$  maximum CSTI of size  $m$  of trees  $R, S$  with polynomial-delay in total time  $\mathcal{O}(|E(R)| \cdot |E(S)| \cdot \max\{|E(R)|, |E(S)|\}^3 \cdot (|V(R)| + 2 - m) + (|V(R)|^2 + |V(S)|^2) \cdot N)$  and total space  $\mathcal{O}((|V(R)| + |V(S)|) \cdot |V(R)| \cdot |V(S)|)$ .*

**Weighted Isomorphisms.** In many application domains graphs are annotated with additional information, e.g., *labels* on edges and vertices. In a graph which is derived from a molecule a vertex label may represent the atom type or charge. If we want to compare such graphs, not only the structure of the graph, but also the similarity between labels must be taken into account. This is true, e.g., for the so-called *feature trees* [15] (cf. Sect. 6). In this section we describe the inclusion of labels into the enumeration of common subtrees.

A *labelled graph* is a graph  $G = (V, E)$  with a label-function  $l : V \cup E \rightarrow \Sigma$ , where  $\Sigma$  is a finite alphabet. To enumerate all maximum CSTIs on two labelled trees  $R, S$  with label-functions  $l_R : V(R) \cup E(R) \rightarrow \Sigma$ ,  $l_S : V(S) \cup E(S) \rightarrow \Sigma$  we apply a symmetric weight-function  $w : \Sigma \times \Sigma \rightarrow \mathbb{Q}$ . Instead of finding common subtree isomorphism with maximum size, we want to enumerate all common subtree isomorphism with maximum weight. We abbreviate this by *maximum CWSTI*. Let  $R' := (V'_R, E'_R)$  and  $S' := (V'_S, E'_S)$  be subtrees of  $R$  and  $S$ , and  $\varphi : V'_R \rightarrow V'_S$  a CSTI of  $R$  and  $S$ . The weight of  $\varphi$  is defined by

$$W(\varphi) := \sum_{v \in V'_R} w(l_R(v), l_S(\varphi(v))) + \sum_{uv \in E'_R} w(l_R(uv), l_S(\varphi(u)\varphi(v))). \quad (1)$$

We describe how to change our enumeration algorithm for maximum CSTIs to handle the enumeration of maximum CWSTIs. First, we describe the necessary modifications for vertices and then for edges. Both modifications consist of modifying the calculation of the table  $D$ . For vertices, we change the table entry  $D(R_s^i, S_t^j) = 1 + M$  (cf. Sect. 2.1) to  $w(l_S(s), l_R(t)) + M$ , i.e., vertices  $s, t$  do not add 1 to the weight (size), but instead add the weight defined by  $w$ .

The changes for edges are similar. We add  $w(l_R(is), l_S(jt))$  to the table entry  $D(R_s^i, S_t^j)$ . Therefore  $D(R_s^i, S_t^j)$  is the weight of a maximum CWSTI including the weight for the mapped edges  $is, jt$ . When we add  $D(R_s^i, S_t^j) + D(R_i^s, S_j^t)$  we add the weight for the “center” edges, i.e.,  $w(l_R(is), l_S(jt))$ , twice. So we have to subtract it to get the weight of a maximum CWSTI  $\varphi$  of  $R$  and  $S$  with  $\varphi(i) = j, \varphi(s) = t$ .

We have to be careful with negative weights for a pair of edges and non-positive weights for a pair of vertices. In this case a maximum CWSTI might not be maximal. The handling of this issue is more complicated and described in [6]. It is obvious that the time complexity of the presented approach for restricted weights remains unchanged compared to the unweighted algorithm.

## 5 Enumeration of Maximal Common Subtree Isomorphisms

There are algorithms that calculate all connected maximal common subgraphs of two input graphs, e.g., by reducing to the enumeration of cliques in a product graph [3], [10]. If we input trees into these algorithms they output all maximal CSTI of the input trees. However, these algorithms take no benefit from the acyclic structure of trees. Also there is yet no other algorithm for enumerating maximal CSTI of given trees. In this section we briefly show how to change our algorithm for maximum CSTI from Section 4 to enumerate maximal CSTI instead. This algorithm is faster than the general algorithm from [3], [10] in theory and practice (cf. Table 1).

The main difference to the enumeration of maximum CSTIs is to enumerate maximal matchings instead of maximum weight matchings in the bipartite graphs given by Edmonds’ algorithm, cf. Sect. 4. However, we cannot apply the edge deletion to enumerate every maximal CSTI only once. Consider the trees  $R = (\{a, b, c\}, \{ab, bc\})$  and  $S = (\{1, 2, 3\}, \{12, 23\})$ . One maximal CSTI is  $\varphi(a) = 1, \varphi(b) = 2, \varphi(c) = 3$ . If we delete  $ab \in E(R)$ , one subsequent CSTI will be  $\varphi'(b) = 2, \varphi'(c) = 3$ , which is obviously not maximal. We solve this by not deleting edges, but by assigning each edge  $e \in E(R)$  a unique natural number  $I(e)$ . Assume we selected  $R_v^u$  as a rooted subtree of  $R$ . We output a maximal CSTI  $\varphi$  iff  $R_v^u$  has not yet been selected and  $I(uv)$  is less than  $I(e)$  for every other edge  $e$  in the graph determined by  $\text{dom}(\varphi)$ . This means, we select one of the total  $2|\varphi| - 2$  identical isomorphisms, cf. Sect. 4, and output only this one.

We briefly discuss the time and space complexity of the approach, see [6] for further details. Let  $m$  be the size of a *maximum* CSTI. Then  $|\varphi| \leq m$  for every maximal CSTI  $\varphi$ . Maximal matchings in a complete bipartite graph can be enumerated in amortized time  $\mathcal{O}(1)$  per matching [6]. Since every matching yields at least one additional node to the currently enumerated CSTI, we have time  $\mathcal{O}(m)$  per enumerated CSTI. With regard to the above paragraph we get amortized time  $\mathcal{O}(m^2)$  per outputted maximal CSTI. The space complexity is  $\mathcal{O}(|V(R)| + |V(S)|)$ . This is determined by the space needed for the enumeration of maximal matchings in complete bipartite graphs.

**Table 1. Maximal CSTI vs. reduction to cliques.** Entries are average values (5% quantile / 95% quantile) over 100 pairs of random trees; table columns are: edges of both trees, number of maximal CSTIs, enumerated maximal CSTIs per second, total time to enumerate all maximal CSTIs, maximal CSTIs per second by clique enumeration, and the ratio between the total running times of the two algorithms; k abbreviates thousand, M million.

Edges	Maximal CSTIs	CSTIs per s in M	Total Time CSTI enum.	Cliques/s	Cl./CSTIs
20	59 (7 / 162) k	10.3 (5.6 / 15.4)	5 (1 / 13) ms	82.2 k	125
25	430 (28 / 1942) k	11.9 (8.7 / 15.8)	35 (3 / 156) ms	34.8 k	342
30	2 713 (146 / 10 738) k	12.0 (9.7 / 15.3)	223 (14 / 913) ms	14.5 k	832
35	27 (1 / 73) M	12.2 (9.4 / 15.9)	1 955 (74 / 7 014) ms	running time > 1d	
40	145 (3 / 515) M	12.2 (9.5 / 15.8)	10 785 (284 / 40 738) ms	running time > 1d	
45	911 (22 / 2 858) M	11.8 (9.4 / 14.9)	75 (2 / 240) s	running time > 1d	

**Theorem 2.** *It is possible to enumerate all  $N$  maximal common subtree isomorphism of trees  $R, S$  in total time  $\mathcal{O}(m^2N)$  and total space  $\mathcal{O}(|V(R)| + |V(S)|)$ , where  $m$  is the size of a maximum CSTI of  $R$  and  $S$ .*

The technique to avoid duplicates by means of a function  $I$  can also be applied for the enumeration of maximum CSTI. With respect to practical running times on not too small random tree instances (cf. Sect. 6), this approach performs similar to the method based on edge deletion. We save the time for the recalculation of the table  $D$ , but we do not output all generated isomorphisms. Note that when using the function  $I$  to select the isomorphisms to output, we do not obtain a polynomial-delay algorithm any more. However, we still have polynomial total time obviously.

We can apply the weighted variant from Sect. 4 to the enumeration of maximal CSTI. However, this does not change the isomorphisms, but yields the weights instead of the size.

## 6 Experimental Evaluation

In this section, we report on our computational results. All experiments were performed on an Intel Core i5 3570K CPU with 8 GB RAM. Both trees  $R$  and  $S$  have the same number of edges if not stated otherwise and are pseudo-randomly generated using the Open Graph Drawing Framework<sup>1</sup>. First, we compare the time to enumerate maximal CSTIs with our new algorithm to the time required by the algorithm in [3, 10] based on enumeration of cliques in the corresponding product graphs. The results are shown in Table 1. As expected, our algorithm clearly beats the more general clique based approach on trees, the relative speedup grows with the size of the input. The second column shows a sublinear running time in practice for the enumeration of all maximal CSTI on random trees.

<sup>1</sup> <http://www.ogdf.net/>

**Table 2. Maximum CSTI.** Notation: cf. Table 1; table columns are: edges of both trees, number of maximum CSTIs, enumerated maximum CSTIs per second, total time to enumerate all maximum CSTIs, the fraction of running time spent on (re)calculating the table  $D$  with Edmonds’ algorithm, and the size of the maximum CSTIs.

Edges	Maximum CSTIs	CSTIs per s in M	Total Time enum.	Edmonds	M. CSTI
40	385 (0.2 / 447) k	1.86 (0.01 / 5.67)	49 (16 / 83) ms	75%	28 (25 / 31)
50	8 616 (4.6 / 5 599) k	3.72 (0.17 / 9.72)	764 (27 / 1 014) ms	53%	34 (30 / 38)
60	14 402 (9.0 / 44 126) k	4.76 (0.22 / 11.40)	1 919 (43 / 9 346) ms	36%	39 (35 / 44)
70	374 (0.1 / 827) M	6.17 (1.17 / 12.08)	50 (0.1 / 218) s	18%	45 (40 / 50)

Next, the results for the enumeration of maximum CSTIs are displayed, cf. Table 2. We can clearly observe that the time to calculate the table  $D$  is significant through all examined tree sizes. An improvement of our basic implementation of the Hungarian method may lead to better results. As expected, the enumeration of maximal CSTI is faster, but for tree instances with many different maximum CSTI the enumeration is not much slower. In our randomly generated trees the average size of the maximum CSTIs is about 2/3 the size of the trees.

Finally, we evaluate our enumeration algorithm for maximum CWSTI on real-world data from cheminformatics. Feature trees [15] are a simplified representation of molecular structures consisting of trees with additional information. We compare the first 1001 feature trees derived from a molecular data set<sup>2</sup> regarding their pairwise similarity. Most trees vary from 3 to 15 vertices. We define a weight function  $w$  with  $\text{im}(w) \subseteq ]0, 1]$  on the vertex labels as described in [15]. Then, we calculate the relative weight of a maximum CWSTI  $\varphi$  of two feature trees  $R$  and  $S$  to the size of the larger input tree, i.e.  $W(\varphi)/\max\{|V(R)|, |V(S)|\}$ . It takes 197 seconds to compute all (1 190 889 in total) maximum CWSTIs between more than half a million pairs of feature trees. Investigation of some of the best matches reveals that their structure is nearly identical. The result shows that our algorithm allows a full analysis considering the ambiguity of maximum CWSTIs even on large real-world data sets.

## 7 Conclusions

We have developed the first efficient algorithms for the enumeration of all maximum, maximal, and maximum weight CSTIs of two given trees. No enumeration algorithms for MCSs with guarantee in terms of running time have been known before. Our approach outputs all maximum and, respectively, maximum weight CSTIs with polynomial-delay and can be modified to list all maximal CSTIs. We as well proposed an algorithm enumerating MaxWBMs with polynomial-delay, which is not only an integral part of our main algorithm, but also of interest in itself. The key ideas of our approach can presumably be taken, too, as a basis

<sup>2</sup> Pyruvate Kinase (AID 361), <http://pubchem.ncbi.nlm.nih.gov/assay/assay.cgi?aid=361>

for finding efficient enumeration algorithms for MCSs in more complex graph classes; this is postponed to future research.

## References

1. Akutsu, T., Tamura, T.: A polynomial-time algorithm for computing the maximum common connected edge subgraph of outerplanar graphs of bounded degree. *Algorithms* **6**(1), 119–135 (2013)
2. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* **16**, 575–577 (1973)
3. Cazals, F., Karande, C.: An algorithm for reporting maximal  $c$ -cliques. *Theoretical Computer Science* **349**(3), 484–490 (2005)
4. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. *Int. J. Pattern. Recognit. Artif. Intell.* **18**(3), 265–298 (2004)
5. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: *Combinatorial Optimization*. John Wiley & Sons Inc., New York (1998)
6. Droschinsky, A.: *Effiziente Enumerationsalgorithmen für Common Subtree Probleme*. Master’s thesis, Technische Universität Dortmund (2014)
7. Fukuda, K., Matsui, T.: Finding all minimum-cost perfect matchings in bipartite graphs. *Networks* **22**(5), 461–468 (1992)
8. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. *Information Processing Letters* **27**(3), 119–123 (1988)
9. Kao, M.Y., Lam, T.W., Sung, W.K., Ting, H.F.: An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *J. Algorithms* **40**(2), 212–233 (2001)
10. Koch, I.: Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science* **250**(12), 1–30 (2001)
11. Kriege, N., Mutzel, P.: Finding maximum common biconnected subgraphs in series-parallel graphs. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) *MFCS 2014, Part II*. LNCS, vol. 8635, pp. 505–516. Springer, Heidelberg (2014)
12. Kuhn, H.W.: The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* **2**, 83–97 (1955)
13. Levi, G.: A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo* **9**(4), 341–352 (1973)
14. Matula, D.W.: Subtree isomorphism in  $O(n^{5/2})$ . In: *Algorithmic Aspects of Combinatorics*, *Ann. Discrete Math.*, vol. 2, pp. 91–106. Elsevier (1978)
15. Rarey, M., Dixon, J.: Feature trees: A new molecular similarity measure based on tree matching. *Journal of Computer-Aided Molecular Design* **12**(5), 471–490 (1998)
16. Raymond, J.W., Willett, P.: Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *J. Comput. Aided Mol. Des.* **16**(7), 521–533 (2002)
17. Schietgat, L., Ramon, J., Bruynooghe, M.: A polynomial-time metric for outerplanar graphs. In: Frasconi, P., Kersting, K., Tsuda, K. (eds.) *Mining and Learning with Graphs* (2007)
18. Uno, T.: Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In: Leong, H.-V., Jain, S., Imai, H. (eds.) *ISAAC 1997*. LNCS, vol. 1350, pp. 92–101. Springer, Heidelberg (1997)
19. Yamaguchi, A., Aoki, K.F., Mamitsuka, H.: Finding the maximum common subgraph of a partial  $k$ -tree and a graph with a polynomially bounded number of spanning trees. *Inf. Process. Lett.* **92**(2), 57–63 (2004)

# Efficient Enumeration of Induced Subtrees in a $K$ -Degenerate Graph

Kunihiro Wasa<sup>1</sup>(✉), Hiroki Arimura<sup>1</sup>, and Takeaki Uno<sup>2</sup>

<sup>1</sup> Graduate School of Information Science and Technology, Hokkaido University,  
Sapporo, Japan

{wasa, arim}@ist.hokudai.ac.jp

<sup>2</sup> National Institute of Informatics, Tokyo, Japan  
uno@nii.jp

**Abstract.** In this paper, we address the problem of enumerating all induced subtrees in an input  $k$ -degenerate graph, where an *induced subtree* is an acyclic and connected induced subgraph. A graph  $G = (V, E)$  is a  $k$ -degenerate graph if for any its induced subgraph has a vertex whose degree is less than or equal to  $k$ , and many real-world graphs have small degeneracies, or very close to small degeneracies. Although, the studies are on subgraphs enumeration, such as trees, paths, and matchings, but the problem addresses the subgraph enumeration, such as enumeration of subgraphs that are trees. Their induced subgraph versions have not been studied well. One of few examples is for chordless paths and cycles. Our motivation is to reduce the time complexity close to  $O(1)$  for each solution. This type of optimal algorithms is proposed many subgraph classes such as trees, and spanning trees. Induced subtrees are fundamental object thus it should be studied deeply and there possibly exist some efficient algorithms. Our algorithm utilizes nice properties of  $k$ -degeneracy to state an effective amortized analysis. As a result, the time complexity is reduced to  $O(k)$  time per induced subtree. The problem is solved in constant time for each in planar graphs, as a corollary.

## 1 Introduction

*Subgraph enumeration problems* are enumeration problems that given a graph  $G$  and a graph class  $\mathcal{S}$ , output all subgraphs  $S$  of  $G$  satisfying  $S \in \mathcal{S}$  without duplicates. Subgraph enumeration problems are widely studied [1–3, 7–11]. Enumeration involves a huge number of solutions, thus enumeration algorithms are supposed to run in short time, with respect to the number of solutions  $N$ . For example, if an algorithm runs in  $O(Nf)$  time for small  $f$ , other than preprocessing, we can consider the algorithm is efficient. In this case, we say that the algorithm runs in  $O(f)$  time per solution, or  $O(f)$  time for each solution. Further, the maximum computation time between two consecutive outputs called *delay* is also considered as a more efficiency of enumeration algorithms. Note that delay will not be  $O(f)$  even if an algorithm runs in  $O(f)$  time per solution.

Enumeration algorithms are widely studied in these days. Especially, the data mining area has a large amount of studies on pattern mining problem. The

algorithms have to deal with huge databases and a huge number of solutions, thus there are great needs of the algorithm theory on efficient enumeration. As we show below, many recent studies focus on the development of small complexity algorithms. Compared to other algorithms, enumeration algorithms have some unique aspects. For example, by operating only on the differences between the solutions, one can develop algorithms that run in time shorter than the amount of exact output. Other than this, since the recursion is much more structured compared to optimization, we can develop a non-trivial amortized analysis. As a consequent, researches on the enumeration algorithms have great interests.

In what follows, we fix the input graph  $G = (V, E)$ , and let  $m = |E|$ ,  $n = |V|$ . In the 1970s, Tarjan and Read [9] studied a problem of enumerating spanning trees in the input graph. Their algorithm runs in  $O(m + n + mN)$  time. Shioura, Tamura, and Uno [7] is improved the complexity to  $O(n + m + N)$  time. Tarjan [8] proposed an algorithm for enumerating all cycle in  $O((|V| + |E|)(|\mathcal{C}(G)| + 1))$  time, where  $\mathcal{C}(G)$  is all cycle in  $G$ . Birmelé *et al.* [2] improved the complexity to in  $O(m + \sum_{c \in \mathcal{C}(G)} |c|)$  total time. They also presented an enumeration algorithm for all st-paths in the input graph  $G$  in  $O(m + \sum_{\pi \in \mathcal{P}_{st}(G)} |\pi|)$  total time, where  $\mathcal{P}_{st}(G)$  is all st-paths in  $G$ . Ferreira *et al.* [3] proposed an enumeration algorithm that enumerating all subtree having exactly  $k$  edges in  $G$  in  $O(kN)$  time. Wasa *et al.* [11] presented an improved version of Ferreira *et al.*'s problem in constant time delay when the input is a tree. As we see, speed up of enumeration algorithms have been intensively studied in long history.

Compared to these studies, induced subgraph enumerations have not been studied well. Avis and Fukuda [1] considered the connected induced subgraph enumeration problem. Their algorithm is based on reverse search, and runs in  $O(mnN)$  time. Uno [10] proposed an enumeration algorithm for enumerating all chordless path connecting the given vertices  $s$  and  $t$  and all chordless cycle in  $O((m + n)N)$  time. Ferreira *et al.* [4] also proposed an enumeration algorithm for this problem. Their algorithm runs in  $\tilde{O}(|n|)$  time per chordless cycle. Their algorithm also enumerates all st-chordless paths with the same complexity.

In this paper, we address the problem of enumerating all induced subtrees in the given graph, where an induced subtree is a connected induced subgraph that has no cycle. Assume that the set of vertices in an induced subtree is  $S$ . Then,  $V \setminus S$  is a feedback vertex set of  $G$ . Feedback vertices are also fundamental graph objects and their enumeration problem is equivalent to that of induced subtrees. If the input graph  $G$  is a tree, the connected induced subgraph of  $G$  is a subtree. Thus, Wasa *et al.*'s shows that the induced subtree enumeration problem can be solved in constant time delay when the input graph is a tree. Tree is a simple graph class, so we are motivated whether we can do better in more general graph classes with non-trivial algorithms.

As a main result of this paper, we propose an algorithm for the  $k$ -degenerate graph case. The algorithm runs in  $O(k)$  time per solution, after  $(|V| + |E|)$  preprocessing time. The algorithm starts from the empty subgraph, and adds a vertex recursively to enlarge the induced subtree. The vertex to be added has to be adjacent to the current induced subtree, and has not to make a cycle.



By using the degeneracy, we efficiently maintain the addible vertices, and the time complexity is bounded by a sophisticated amortized analysis. Real world graphs usually have small degeneracies, or only few vertex removals result small degeneracies, the algorithm is expected to be efficient in practice. Compared to other graph classes, this is a strong point of  $k$ -degenerate graphs. There have been not so many studies on the use of the degeneracy for enumeration algorithm, and thus our approach introduces one of new way of developing practically efficient and theoretically supported algorithms.

The rest of this paper is organized as follows: In Section 2, we gives definitions in this paper and the definition of our problem. In Section 3, we propose a basic enumeration algorithm based on a binary partition method. In Section 4, we improve the algorithm by using a property of the degeneracy, and analyze its time complexity. Finally, we conclude this paper and give future works in Section 5.

## 2 Preliminaries

### 2.1 Graphs

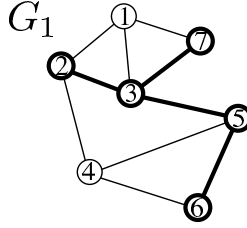
Let  $G = (V, E)$  be an *undirected graph*, where  $V$  is the set of *vertices* and  $E \subseteq V^2$  is the set of *edges*. In this paper, we assume that  $G$  is simple and finite. We denote by  $(u, v)$  the edge connecting  $u$  and  $v$ . For any vertices  $u, v$  of  $V$ , we say that  $u$  and  $v$  are *adjacent* to each other if  $(u, v) \in E$ . We denote by  $N_G(u)$  the set of all vertices adjacent to  $u$  in  $G$ . We define the *degree*  $d_G(u)$  of  $u$  in  $V$  as the number of vertices adjacent to  $u$ . In what follows, if it is clear from context, we omit the subscript  $G$ .

A *path* in  $G$  is a sequence of distinct vertices  $\pi(u, v) = (v_1 = u, \dots, v_j = v)$ , such that  $v_i$  and  $v_{i+1}$  are adjacent to each other for  $1 \leq i < j$ . If there is  $\pi(u, v)$  in  $G$ , we say that the path *connects*  $u$  and  $v$ . The *length* of path  $\pi(u, v)$  is the number of vertices in  $\pi(u, v)$  minus one. For any path  $\pi(u, v)$  of length larger than one,  $\pi(u, v)$  is called a *cycle* if  $u = v$ . We say that  $G$  is *connected* if there is a path connecting any pair of vertices in  $G$ .  $G$  is a *tree* if  $G$  has no cycle and is connected.

### 2.2 Induced Subtrees

Let  $S$  be a subset of  $V$ . We denote by  $G[S] = (S, E[S])$  the graph *induced* by  $S$ , where  $E[S] = \{(u, v) \in E \mid u, v \in S\}$ . We call  $G[S]$  an *induced subgraph* of  $G$ . If no confusion, we regard  $S$  as  $G[S]$ .  $|S|$  is the size of  $S$ . We say that  $S$  is an *induced subtree* (see Fig. 1), if  $S$  is a tree. In the following, we state the problem of this paper.

*Problem (Induced subtree enumeration problem).* Enumerate all induced subtrees in  $G = (V, E)$ .



**Fig. 1.** An induced subtree  $S_1$  in  $G_1$ . In the figure, bolded vertices and edges represent vertices and edges in  $S_1$ .  $S_1$  consists of  $\{2, 3, 5, 6, 7\}$ .  $S_1$  is an induced subtree in  $G_1$  since  $S_1$  is connected and acyclic.

### 2.3 $K$ -Degenerate Graphs

A graph  $G$  is  $k$ -degenerate [5] if any its induced subgraph of  $G$  has a vertex whose degree is less than or equal to  $k$ . The *degeneracy* of  $G$  is defined as the smallest  $k$  satisfying the definition of  $k$ -degenerate graphs. Examples of graph classes with constant degeneracy include trees, grid graphs, outerplanar graphs, and planar graphs, thus degenerate graph is a large class of sparse graphs. These degeneracy are 1, 2, 2, and 5, respectively.

From the definition of  $k$ -degeneracy, we obtain a vertex sequence  $(u_1, \dots, u_{|V|})$  satisfying the condition

$$\forall 1 \leq i \leq |V|, |\{u_j \in N(u_i) \mid i < j \leq |V|\}| \leq k \cdots (\star).$$

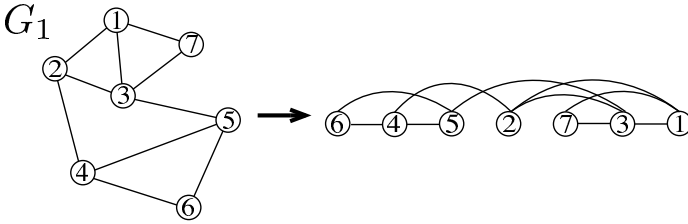
This condition  $(\star)$  implies that there exists an *ordering* among vertices of  $G$  such that for any vertex  $u$ , the number of vertices adjacent to  $u$  larger than it is at most  $k$ . Hereafter we assume that the vertices are indexed in this ordering. We say  $u < v$  ( $u > v$ , respectively) if the index of  $u$  is smaller than  $v$  ( $u$  is larger than  $v$ , respectively) with respect to this ordering. In Fig. 2, we show an example of the ordering satisfying  $(\star)$ . Matula and Beck [6] proposed an algorithm for obtaining the degeneracy of  $G$  and the ordering satisfying  $(\star)$ . By iteratively choosing the smallest degree vertex and removing it from  $G$ , their algorithm finds such an ordering in  $O(|V| + |E|)$  time.

## 3 Basic Binary Partition Algorithm

### 3.1 Candidate Sets and Forbidden Sets

Let  $S$  be an induced subtree of  $G$ . We define the *adjacency* of a vertex  $u \in V$  to  $S$  as  $\text{adj}(S, u) = |S \cap N(u)|$ , that is,  $\text{adj}(S, u)$  is the number of vertices of  $S$  adjacent to  $u$ .

**Lemma 1.** *Let  $S$  be any induced subtree in  $G$  and  $u$  be any vertex  $V \setminus S$ .  $S \cup \{u\}$  is an induced subtree if and only if  $\text{adj}(S, u) = 1$ .*



**Fig. 2.** An example of an ordering of  $G_1 = (V_1, E_1)$ . In the right graph, vertices are sorted by the ordering that satisfies  $(\star)$ .

*Proof.* If  $\text{adj}(S, u) > 1$ ,  $u$  is adjacent to two vertices  $v$  and  $w$  of  $S$ . Since  $S$  has a path  $\pi$  connecting  $v$  and  $w$ , the addition of  $u$  yields a cycle in  $S \cup \{u\}$ . If  $\text{adj}(S, u) = 0$ ,  $S \cup \{u\}$  is disconnected. If  $\text{adj}(S, u) = 1$ ,  $S \cup \{u\}$  is connected. Since the degree of  $u$  in  $G[S \cup \{u\}]$  is one,  $u$  is not included in a cycle. Thus,  $G[S \cup \{u\}]$  does not contain a cycle.  $\square$

In each iteration, we maintain the *forbidden set*  $X$  as the vertex set such that any vertex  $u$  in  $X$  satisfies either  $u$  belongs to  $S$ ,  $S \cup \{u\}$  includes a cycle, or  $u$  is forbidden to include in the solution by some ancestor iterations of the iteration. We also maintain the *candidate set*  $CAND$  as the set of vertices whose additions yield induced subtrees and are not included in  $X$ . We maintain  $CAND$  and  $X$  for efficient computation. From Lemma 1, they are disjoint, and for any vertex  $u$ , if  $\text{adj}(S, u) > 0$ ,  $u$  belongs to either  $CAND$  or  $X$ .

### 3.2 Basic Binary Partition

Our algorithm starts from the empty induced subtree  $S = \emptyset$ . In each iteration given an induced subtree  $S$ , we remove a vertex  $u$  from  $CAND$ , and partition the problem into two; enumeration of all induced subtrees including  $S \cup \{u\}$ , and those including  $S$  but not including  $u$ . We recursively do this partition until there is no vertex in  $CAND$ . The former can be solved by a recursive call with setting  $S$  to  $S \cup \{u\}$ . The latter is solved by a recursive call with setting  $X$  to  $X \cup \{u\}$ . In this way, we can enumerate all induced subtrees. We present the main routine ISE of our algorithm in Algorithm 1. We show how to update candidate sets and forbidden sets in the next two lemmas.

**Lemma 2.** For an induced subtree  $S$  and a vertex  $u \in CAND$ , when we add  $u$  to  $S$  and remove  $u$  from  $CAND$ ,  $CAND$  changes to

$$(CAND \setminus N(u)) \cup (N(u) \setminus (CAND \cup X)).$$

*Proof.* Any vertex in  $CAND$  other than  $N(u)$  remains in  $CAND$  after the addition of  $u$  to  $S$  since the adjacencies of the vertices do not change. If vertices in  $N(u) \cap (CAND \cup X)$  are added to  $S \cup \{u\}$ , then they are in  $S$ , they are forbidden

---

**Algorithm 1.** Main routine ISE: Enumerating all induced subtrees in  $G$

---

- 1: **procedure** ISE( $G = (V, E), S, CAND, X$ )
  - 2:   **if**  $CAND = \emptyset$  **then** output  $S$ ; **return**;
  - 3:   choose the smallest vertex  $u$  from  $CAND$  and remove  $u$  from  $CAND$ ;
  - 4:   call ISE( $G, S, CAND, X \cup \{u\}$ );
  - 5:   call ISE( $G, S \cup \{u\}, (CAND \setminus N(u)) \cup (N(u) \setminus CAND), X \cup \{u\} \cup (CAND \cap N(u))$ );
- 

to be add to  $S$  and its decendants, or they make cycles since they are adjacent to  $u$  and other vertices in  $S$ . The adjacency of any vertex in  $N(u) \setminus (CAND \cup X)$  is zero for  $S$ , and one for  $S \cup \{u\}$ . Any vertex  $v \notin S$  satisfying  $\text{adj}(S \cup \{u\}, v) = 1$  is either in  $N(u)$  or  $CAND$ . Thus, the statement holds.  $\square$

**Lemma 3.** For an induced subtree  $S$  and a vertex  $u \in CAND$ , when we add  $u$  to  $S$  and remove  $u$  from  $CAND$ ,  $X$  changes to

$$X \cup \{u\} \cup (CAND \cap N(u)).$$

*Proof.* Any vertex  $v \in X$  remains in  $X$  for  $S \cup \{u\}$ , since  $\text{adj}(S \cup \{u\}, v) \geq \text{adj}(S, v)$  always holds. From the definition of the forbidden set,  $u$  is in  $X$  for  $S \cup \{u\}$ . Further, any vertex  $v$  in  $CAND \cap N(u)$  makes cycles when they are added to  $S \cup \{u\}$ , since  $\text{adj}(S \cup \{u\}, v) \geq 2$  holds. By adding  $u$  to  $S$ , no other vertex is forbidden to be added, thus the statement holds.  $\square$

**Theorem 1.** Algorithm ISE enumerates all induced subtrees in the input graph  $G = (V, E)$  without duplicates.

## 4 Improved Binary Partition Algorithm

From Lemma 2 and Lemma 3, we can easily see that the computation time of updating the candidate set and the forbidden set is  $O(d_G(u))$  by checking all vertices adjacent to  $u$ . However, in this way, we must check some vertices again and again. Specifically, let us assume  $u$  and  $v$  are consecutively added to  $S$ , and  $w \notin S$  is adjacent to  $u$ ,  $v$  and another vertex in  $S$ . When we add  $u$  to  $S$ , we check whether we can add  $w$  to the candidate set of  $S \cup \{u\}$ . After generating  $S \cup \{u\}$ , we check  $w$  again when we add  $v$  to  $S \cup \{u\}$ . In order to avoid this redundant checking, we improve the way of updating the candidate set and the forbidden set by using the following set.

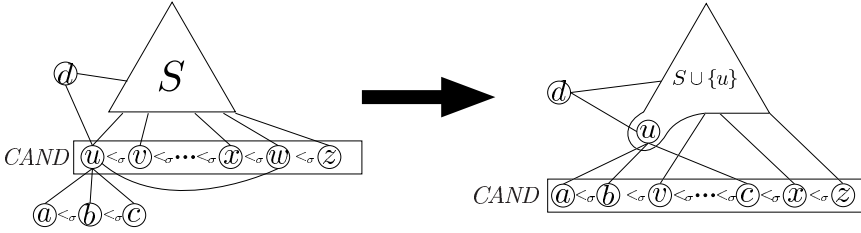
**Definition 1.** Suppose that  $u$  is a vertex of  $CAND$  for an induced subtree of  $G$ . We define a set  $\Gamma(u, X)$  as follows:

$$\Gamma(u, X) = \{v \in N(u) \mid v \notin X, v < u\}.$$

**Lemma 4.** Let  $S$  be an induced subtree of  $G$ ,  $u$  be the smallest in the candidate set  $CAND$  of  $S$ , and  $X$  be the forbidden set of  $S$ . Then, the following formula holds:

$$N(u) \setminus (CAND \cup X) = (N_\ell(u) \setminus (CAND \cup X)) \cup \Gamma(u, X),$$

where  $N_\ell(u) = \{v \in N(u) \mid u < v\}$ .



**Fig. 3.** This figure shows the changes between candidate set  $CAND$  by the addition of  $u$  to  $S$ .  $S$  is an induced subtree and  $\{u, v, \dots, x, w, z\}$  is the candidate set of  $S$ . Let assume that  $a < b < u < c$  and  $d < u$ . Since  $d$  does not belongs to  $\Gamma(u, X)$ ,  $d$  is skipped checking.

*Proof.* Let  $Z$  be the set of vertices larger than  $u$ . Since  $u$  is the smallest vertex in  $CAND$ ,  $(N(u) \setminus (CAND \cup X)) \cap Z = (N_\ell(u) \setminus (CAND \cup X))$ . From the definition of  $\Gamma(u, X)$  and  $u$  is the smallest in  $CAND$ ,  $(N(u) \setminus (CAND \cup X)) \cap (V \setminus Z) = N_s(u) \setminus (CAND \cup X) = (N_s(u) \setminus CAND) \cap (N_s(u) \setminus X) = \Gamma(u, X)$ , where  $N_s(u) = \{v \in N(u) \mid v < u\}$ . This concludes the lemma.  $\square$

In what follows, we implement  $CAND$ ,  $X$ , and  $\Gamma$  by doubly linked lists. Thanks to the doubly linked list, the cost for a removal and the recover of the removed element can be done in constant time, and the merge of two sets can be done in linear time of the sum of their sizes. In each iteration, we keep vertices of each list sorted in the ordering that satisfies  $(\star)$ .

**Lemma 5.** *When we add a vertex  $u$  to  $X$ , the update of  $\Gamma(v, X)$  for all vertices  $v$  is done in  $O(k)$  time.*

*Proof.* To update, it is suffice to remove  $u$  from  $\Gamma(v, X)$  from all  $v > u$ . Thus, it takes  $O(k)$  time.  $\square$

**Lemma 6.** *Let  $S$  be an induced subtree of  $G$ ,  $u$  be the smallest in the candidate set  $CAND$  of  $S$ , and  $X$  be the forbidden set of  $S$ . When we add  $u$  to  $S$  and remove  $u$  from  $CAND$ , the computation time of updating  $CAND$  and  $X$  are  $O(k + |\Gamma(u, X)|)$  and  $O(k)$  time, respectively.*

*Proof.* Since  $u$  is the smallest vertex in  $CAND$ ,  $|\Delta| \leq k$ , where  $\Delta = |CAND \cap N(u)|$ . Since vertices in  $N(u)$  are sorted by the ordering, the computation time of  $\Delta$  is  $O(k)$ . Thus, adding vertices in  $\Delta$  and  $u$  to  $X$  and removing  $\Delta$  from  $CAND$  are done in  $O(k)$  time. From Lemma 4, since  $|\{v \in N(u) \mid u < v\}| \leq k$ , the computation time of adding these vertex to  $CAND$  is  $O(k + |\Gamma(u, X)|)$ . Hence, the lemma holds.  $\square$

In Fig. 3, we show the changes of between the candidate set of  $S$  and that of  $S \cup \{u\}$  after adding  $u$  to  $S$ .

**Theorem 2.** *Let  $G = (V, E)$  be the input graph and  $k$  is the degeneracy of  $G$ . Our algorithm enumerates all induced subtrees in  $G$  in  $O(k)$  time per solution after  $O(|V| + |E|)$  preprocessing time without duplicates using  $O(|V| + |E|)$  space.*

*Proof.* Since the update of  $CAND$  and  $X$  is correct, the correctness of the algorithm is obvious. (I) We discuss the time complexity of the preprocessing. First, our algorithm computes an ordering of vertices by Matula and Beck's algorithm [6] in  $O(|V| + |E|)$  time. Next, our algorithm sorts vertices belonging to each adjacency list by using a bucket sort. Thus, the preprocessing time is  $O(|V| + |E|)$ .

(II) We consider an iteration inputting  $S$ ,  $X$ , and  $CAND$ , and assume that  $CAND'$  is the candidate set for  $S \cup \{u\}$ . Line 2 and line 3 run in  $O(1)$  time. From Lemma 5, line 4 needs  $O(k)$  time. From Lemma 6, since it is clear that  $|\Gamma(u, X)| \leq |CAND'|$ , our algorithm needs  $O(k + |CAND'|)$  time for computing  $CAND'$ . The update of  $\Gamma$ 's is done in  $O(k|CAND \cap N(u)|)$  time, from Lemma 5. We observe that for each vertex  $w$  such that  $v \in CAND \cap N(u)$  is removed from  $\Gamma(w, X)$ ,  $w$  is in  $CAND$  of  $S \cup \{v\}$ , that will be generated by a descendant of this iteration. We charge the cost of constant time to remove  $v$  from  $\Gamma(w, X)$  to the induced subtree  $S \cup \{v, w\}$ . Then, we can see that  $S \cup \{v, w\}$  is charged only from iterations inputting  $S$ , that divides the problem by  $u'$  such that  $(u', v) \in E$ , that is, the iteration generates  $S \cup \{u'\}$ . We consider the average amount of the charge over all induced subtrees of  $S \cup \{v, w\}$ ,  $v \in CAND$ , and  $w$  is in  $CAND$  of  $S \cup \{v\}$ . Since the number of pairs  $\{u, v\} \subseteq CAND$  is at most  $k|CAND|$ , we can see the average charge is  $O(k)$  for each  $S \cup \{v, w\}$ . Thus, in summary, we can see the update time for  $\Gamma$  in an iteration is bounded by  $O(k)$ , on average. Thus, an iteration takes  $O(k + |CAND'|)$  time on average. We observe that the sum of  $|CAND'|$  over all iterations is no greater than the sum of  $|CAND|$  over all induced subtrees, since  $CAND'$  is the candidate set of  $S \cup \{u\}$  and forbidden set  $X \cup \{u\}$ , and  $S \cup \{u\}$  is generated only from  $S$ . Further, we can see that  $S \cup \{u\}$  is generated only from  $S$  this iteration. Hence, thus the sum of  $|CAND|$  over all induced subtrees is bounded by the number of induced subtrees. Therefore, the computation time for each iteration is bounded by  $O(k)$  on average.

In a binary partition algorithm, each iteration at the leaf of the recursion outputs a solution, and each non-leaf iteration generates exactly two recursive calls. Thus, the number of iterations (recursive calls) of a binary partition algorithm is at most  $2N$ . Hence, the computation time per induced subtree is  $O(k)$ . All sets the algorithm maintains are of size  $O(|V| + |E|)$  in total.

We need a bit care to perform a recursive call. When a recursive call is made, we record the operations to prepare the parameters given to the recursive call on the memory. When the recursive call ends, we apply the inverse operations of the recorded operations to recover the variables such as  $CAND$  and  $X$ . In this way, we can recover the variables from the updated ones without increasing the time complexity. Since no vertex is added or deleted from the same variable twice, the accumulated space for the recorded operations is bounded by  $O(|V| + |E|)$ . From the above arguments, our algorithm runs in  $O(k)$  time per solution after  $O(|V| + |E|)$  preprocessing time using  $O(|V| + |E|)$  space.  $\square$

## 5 Conclusion

In this paper, we have presented an algorithm for enumerating all induced subtrees in  $k$ -degenerate graph. Our algorithm runs in  $O(k)$  time per solution after linear preprocessing time using linear space. From this result, we obtain the following corollary; if the input graph has a constant degeneracy, our algorithm is optimal with respect to the computation time per solution.  $K$ -degenerate graphs often appear in real-world data even when with much noise. Thus considering the applications, it is important to study on efficient computation on  $k$ -degeneracy. This result is one of the first steps for such studies, and researches on enumeration algorithms on  $k$ -degenerate graphs will be an important issue.

**Acknowledgments.** This work was partially supported by MEXT Grant-in-Aid for Scientific Research (A) 24240021 and Grant-in-Aid for JSPS Fellows 25 · 1149.

## References

1. Avis, D., Fukuda, K.: Reverse search for enumeration. *DAM* **65**, 21–46 (1996)
2. Birmelé, E., Ferreira, R.A., Grossi, R., Marino, A., Pisanti, N., Rizzi, R., Sacomoto, G.: Optimal Listing of Cycles and st-Paths in Undirected Graphs. In: Proc. SODA 2013, pp. 1884–1896 (2013)
3. Ferreira, R., Grossi, R., Rizzi, R.: Output-sensitive listing of bounded-size trees in undirected graphs. In: Demetrescu, C., Halldórsson, M.M. (eds.) *ESA 2011*. LNCS, vol. 6942, pp. 275–286. Springer, Heidelberg (2011)
4. Ferreira, R., Grossi, R., Rizzi, R., Sacomoto, G., Sagot, M.-F.: Amortized  $\tilde{O}(|V|)$ -Delay Algorithm for Listing Chordless Cycles in Undirected Graphs. In: Schulz, A.S., Wagner, D. (eds.) *ESA 2014*. LNCS, vol. 8737, pp. 418–429. Springer, Heidelberg (2014)
5. Lick, D.R., White, A.T.:  $k$ -degenerate graphs. *Can. J. Math.* **XXII**(5), 1082–1096 (1970)
6. Matula, D.W., Beck, L.L.: Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM* **30**(3), 417–427 (1983)
7. Shioura, A., Tamura, A., Uno, T.: An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.* **26**(3), 678–692 (1997)
8. Tarjan, R.E.: Enumeration of the Elementary Circuits of a Directed Graph. *SIAM J. Comput.* **2**(3), 211–216 (1973)
9. Tarjan, R.E., Read, R.C.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks* **5**(3), 237–252 (1975)
10. Uno, T.: An output linear time algorithm for enumerating chordless cycles. Technical Notes, 92nd SIGAL of IPSJ, pp. 47–53 (2003) (in Japanese)
11. Wasa, K., Kaneta, Y., Uno, T., Arimura, H.: Constant time enumeration of bounded-size subtrees in trees and its application. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) *COCOON 2012*. LNCS, vol. 7434, pp. 347–359. Springer, Heidelberg (2012)

# An Efficient Method for Indexing All Topological Orders of a Directed Graph

Yuma Inoue<sup>1(✉)</sup> and Shin-ichi Minato<sup>1,2</sup>

<sup>1</sup> Graduate School of Information Science and Technology, Hokkaido University,  
Sapporo-shi, Japan

yuma@ist.hokudai.ac.jp

<sup>2</sup> JST ERATO MINATO Discrete Structure Manipulation System Project,  
Sapporo-shi, Japan

**Abstract.** Topological orders of a directed graph are an important concept of graph algorithms. The generation of topological orders is useful for designing graph algorithms and solving scheduling problems. In this paper, we generate and index all topological orders of a given graph. Since topological orders are permutations of vertices, we can use the data structure  $\pi$ DD, which generates and indexes a set of permutations. In this paper, we propose *Rot- $\pi$ DDs*, which are a variation of  $\pi$ DDs based on a different interpretation. Compression ratios of *Rot- $\pi$ DDs* for representing topological orders are theoretically improved from the original  $\pi$ DDs. We propose an efficient method for constructing a *Rot- $\pi$ DD* based on dynamic programming approach. Computational experiments show the amazing efficiencies of a *Rot- $\pi$ DD*: a *Rot- $\pi$ DD* for  $3.7 \times 10^{41}$  topological orders has only  $2.2 \times 10^7$  nodes and is constructed in 36 seconds. In addition, the indexed structure of a *Rot- $\pi$ DD* allows us to fast post-process operations such as edge addition and random samplings.

**Keywords:** Topological orders · Linear extensions · Permutations · Decision diagrams · Enumerating algorithms · Experimental algorithms

## 1 Introduction

Topological sort is one of the classical and important concepts of graph algorithms. Vertex orders obtained by topological sort are used to analyze characteristics of a directed graph structure and support graph based algorithms [6]. Furthermore, topological orders are equivalent to linear extensions of a poset, i.e., total orders which are in no contradiction with the partially ordered set defined by directed edges of a graph. Thus, topological sort plays an important role in several research areas such as discrete mathematics and computer science, and has many applications such as graph problems and scheduling problems [14].

Linear time algorithms calculating a topological order are classical and well-known algorithms, and dealt with by Cormen et al. [6]. In recent researches, two derived problems are mainly discussed. One of these is an online topological sort, i.e., calculation of a topological order on a dynamic graph. Bender et al. [2]



proposed a topological sort algorithm which allows edge insertions, and Pearce et al. [13] proposed an algorithm which can also handle edge deletions. Another one is the enumeration problem of all topological orders. Ono et al. [12] presented a worst case constant delay time generating algorithm using family trees. The complexity of the counting problem has been studied from several aspects since Brightwell et al. [3] proved that it is  $\#P$ -complete. Bubley et al. [4] proposed a randomized algorithm to approximate the number of all linear extensions. Li et al. [10] provided an experimentally fast algorithm counting all topological orders based on Divide & Conquer method. There are many polynomial time counting algorithms when we restrict the graph structure or fix some graph parameters, e.g., trees and bounded poset width [1, 5].

In this paper, we deal with both of these problems. That is, our goal is generation of all topological orders of given graphs and manipulation of these orders when the graph is dynamically changed, e.g., edge addition. In addition, we implicitly store all topological orders as a compressed data structure in order to handle graphs that are as large as possible. Experimental results, which will be described later, show that our algorithm and data structure work very well:  $3.7 \times 10^{41}$  topological orders of a directed graph with 50 vertices are generated in 36 seconds, and the compressed data size is only about 1 gigabyte. Furthermore, an edge addition query for a directed graph with 25 vertices is done in 1 second.

Our method is based on an indexed data structure compactly representing a set of permutations, *permutation decision diagram*, also called  $\pi DD$  or  $PiDD$  [11]. Although a  $\pi DD$  can be used to achieve our purpose, compression ratio and query processing are not efficient enough practically or theoretically. Thus, we developed a new variation of  $\pi DD$ , named *Rot- $\pi DD$*  (*Rotation-based  $\pi DD$* ). The key idea of our modification is a direct construction of a decision diagram based on the dynamic programming approach. This modification realizes the practical efficiency of compression and query processing, which are also bounded theoretically.

Our contributions in this paper are summarized as follows.

- We provide the first algorithm for implicit generation of all topological orders with dynamic manipulation.
- Time and space for construction and query processing of our algorithm are efficient experimentally and theoretically, while it is difficult to estimate the size and computation time of decision diagrams in general.

The rest of this paper is organized as follows. Section 2 introduces a precise definition of topological sort and algorithms for counting, which will be used in our algorithm. Section 3 introduces  $\pi DD$ s and our modified version (Rot- $\pi DD$ s) for generation of all topological orders. Our algorithm for construction of a Rot- $\pi DD$  is also presented in Section 3. In Section 4, we prove the theoretical bound of the time complexity of our algorithm and the size of the new permutation decision diagram for all topological orders. Section 5 presents experimental results of generation and query processing, comparing with existing  $\pi DD$  and other existing methods. Section 6 gives some consequences of this paper.

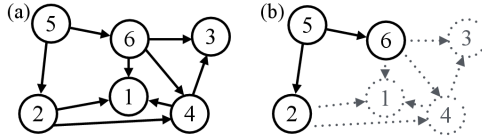


Fig. 1. (a) A DAG and (b) the subgraph induced by the vertex set  $\{2, 5, 6\}$

## 2 Topological Orders

We define a *directed graph*  $G = (V, E)$ , where  $V$  is a vertex set and  $E$  is a directed edge, i.e.,  $E \subseteq \{(u, v) \mid u, v \in V\}$ . Note that  $(u, v)$  is an ordered pair of two vertices. Let  $n$  be the number of vertices and  $m$  be the number of edges. Without loss of generality, we can assume  $V = \{1, 2, \dots, n\}$ .

A *topological order* of a graph  $G$  is an ordering  $v_1v_2 \dots v_n$  of all vertices such that  $v_i$  must precede  $v_j$  if  $(v_i, v_j) \in E$ . For example, the graph in Fig. 1(a) has four topological orders: 526413, 526431, 562413, and 562431.

A directed graph is a *DAG* (*Directed Acyclic Graph*) if the graph has no cycle. In this paper, we assume that given graphs are DAGs because we can determine whether or not a graph has cycles in linear time, and if so, there is no topological order.

There are many linear time algorithms for computing a topological order of a given graph [9, 15]. One of the key ideas is deleting vertices whose out-degree is 0. If there is no edge from  $v$ ,  $v$  can be the rightmost element in a topological order, because there is no element that must be preceded by  $v$ . We delete such  $v$  and its incident edges, i.e., after the deletion of  $v$ , we can consider only the subgraph induced by the vertex subset  $V \setminus \{v\}$ . Then, we repeat the same procedure for the induced subgraph and obtain a topological order of the induced subgraph recursively. Finally, we concatenate a topological order of the induced subgraph and  $v$  to obtain a topological order of the given graph. The time complexity of this algorithm is  $O(n + m)$ .

Similarly, an algorithm counting all topological orders of a given graph can be designed recursively. Let  $G(X)$  denote the subgraph of  $G$  induced by the vertex subset  $X$ . For each recursion, we assume that the current vertex subset is  $V'$ . Then, for each vertex  $v$  whose out-degree is 0 in  $G(V')$ , we sum up the numbers of all topological orders of  $G(V' \setminus \{v\})$ . The time complexity of this algorithm is  $O((n + m)TO(G))$ , where  $TO(G)$  is the number of the topological orders of the graph  $G$ . Since  $TO(G) = O(n!)$ , the time complexity is  $O((n + m)n!)$ . We can improve this complexity by a *dynamic programming* (DP) approach.

For example, in Fig. 1(a), we can delete vertices  $\{1, 3, 4\}$  in the order 134 or 314. (Note that a deletion order is the reverse of a topological order.) Then we obtain the same induced subgraph on  $\{2, 5, 6\}$ . Although  $TO(G(\{2, 5, 6\}))$  is not changed, we redundantly count  $TO(G(\{2, 5, 6\}))$  in each recursion of 134 and 314. Thus, by memorizing the calculation result  $TO(G(V'))$  for  $G(V')$  at the first calculation, we can avoid duplicated calculations for each  $G(V')$ . In

other words, this is a top-down DP, which recursively calculates  $TO(G(V')) = \sum_{v \in V'_0} TO(G(V' \setminus \{v\}))$ , where  $V'_0$  is the set of vertices whose out-degree is 0 in  $G(V')$ . We define *valid induced subgraphs* of  $G$  as induced subgraphs  $G(V')$  that can appear in the above DP recursion. Let  $IS(G)$  denote the number of valid induced subgraphs of  $G$ . Then, this DP algorithm uses  $O((n+m)IS(G))$  time and  $O(IS(G))$  space. In the worst case,  $IS(G) = 2^n$ , which is the number of all subsets of  $V$ . Therefore, we improve the complexity from factorial  $O((n+m)n!)$  to exponential  $O((n+m)2^n)$ .

The idea of valid induced subgraphs is equivalent to upsets in a poset in the talk of Cooper [5]. Cooper provided another upper bound  $O(n^w)$  of  $IS(G)$ , where  $w$  is the width of a poset corresponding to  $G$ . The proof of this bound and more precise analyses will be described in Section 4.

Here, we remember our goal in this paper again. Our goal is generating and indexing all topological orders, which are permutations of vertices. Thus, it is reasonable to expect that a compressed and indexed data structure for permutations can be useful for this purpose. And if we can compress permutations in the same way as the above DP, the compression size is bounded by  $IS(G) = O(\min\{2^n, n^w\})$ , which can be quite smaller than  $TO(G)$ .

### 3 Permutation Decision Diagrams

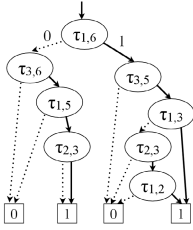
In this section, we introduce a compressed and indexed data structure for permutations,  $\pi$ DD, and discuss whether or not compression of a  $\pi$ DD is suitable for the DP approach.

#### 3.1 Existing Permutation Decision Diagrams: $\pi$ DDs

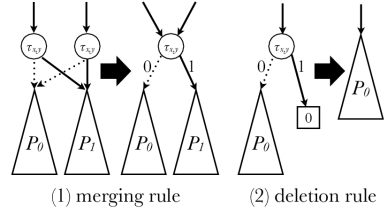
First, we define some notations about permutations. A *permutation* of length  $n$ , or  $n$ -permutation, is a numerical sequence  $\pi = \pi_1\pi_2 \dots \pi_n$  such that all elements are distinct and  $\pi_i \in \{1, 2, \dots, n\}$  for each  $i$ . The identity permutation of length  $n$  is denoted by  $e^n$ , which satisfies  $e_i^n = i$  for each  $1 \leq i \leq n$ .

We define a *swap*  $\tau_{i,j}$  as the exchange of the  $i$ th element and the  $j$ th element. Any  $n$ -permutation can be uniquely decomposed into a sequence of at most  $n-1$  swaps. This swap sequence is defined as the series of swaps to obtain an objective  $n$ -permutation  $\pi$  from the identity permutation  $e^n$  by a certain algorithm. The algorithm repeats swaps to move  $\pi_k$  to the  $k$ th position, where  $k$  runs from right to left. For example, we consider a decomposition of the permutation  $\pi = 43152$  into a swap sequence. We start with  $e^5 = 12345$ . The 5th element of  $\pi$  is 2 and 2 is the 2nd element of  $e^n$ , hence we swap the 2nd element and the 5th element, and obtain  $15342 = \tau_{2,5}$ . Next, since the 4th element of  $\pi$  is 5, and 5 is the 2nd element, we then obtain  $14352 = \tau_{2,5} \cdot \tau_{2,4}$ . Repeating this procedure, we finally obtain  $\pi = 43152 = \tau_{2,5} \cdot \tau_{2,4} \cdot \tau_{1,3} \cdot \tau_{1,2}$ .

A  $\pi$ DD is a data structure representing a set of permutations canonically [11], and has efficient set operations for permutation sets.  $\pi$ DDs consist of five components: nodes with a swap label, 0-edges, 1-edges, the 0-sink, and the 1-sink. Fig. 2 shows the  $\pi$ DD representing topological orders of the graph in Fig. 1(a).



**Fig. 2.** The  $\pi$ DD representing  $\{526413, 526431, 562413, 562431\}$



**Fig. 3.** Two reduction rules of  $\pi$ DDs

Each internal node has exactly a 0-edge and a 1-edge. Each path in a  $\pi$ DD represents a permutation: if a 1-edge originates from a node with label  $\tau_{x,y}$ , the decomposition of the permutation contains  $\tau_{x,y}$ , while a 0-edge from  $\tau_{x,y}$  means that the decomposition excludes  $\tau_{x,y}$ . If a path reaches the 1-sink, the permutation corresponding to the path is in the set represented by the  $\pi$ DD. On the other hand, if a path reaches the 0-sink, the permutation is not in the set.

A  $\pi$ DD becomes a compact and canonical form by applying the following two reduction rules (Fig. 3):

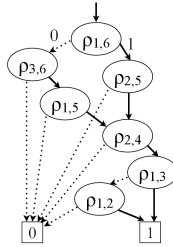
- (1) Merging rule: share all nodes which have the same labels and child nodes.
- (2) Deletion rule: delete all nodes whose 1-edge points to the 0-sink.

Although the size of a  $\pi$ DD (i.e. the number of nodes in a  $\pi$ DD) can grow exponentially ( $O(2^{n^2})$ ) with respect to the length of permutations, in many practical cases,  $\pi$ DDs demonstrate high compression ratio. In addition,  $\pi$ DDs support efficient set operations such as union, intersection, and set difference. The computation time of  $\pi$ DD operations depends on the size of  $\pi$ DDs, not on the number of permutations in the sets represented by the  $\pi$ DDs.

### 3.2 DP Approach and $\pi$ DDs

Now, we consider whether or not we can directly construct a  $\pi$ DD in the same way as the DP approach described in Section 2.

Here, we note that the swap decomposition algorithm behaves as deletions of a vertex on an induced subgraph. We can represent the current recursive state in DP procedure as a permutation, i.e., let  $k$  be the number of vertices of the current induced subgraph, then the  $k$ -prefix of an  $n$ -permutation represents the vertex set of the induced subgraph, and the  $(n - k)$ -suffix of the permutation represents the reverse order of deletions. Furthermore, a deletion of a vertex  $v$  can be described as a swap  $\tau_{i,k}$ , where  $i$  is the position of  $v$  in the permutation. For example, we can consider a permutation 625431 represents the subgraph in Fig. 1(b) such that the deletion order is 134. When we delete the vertex 6, we swap the 1st position, which is 6, and the 3rd position, which is the rightmost



**Fig. 4.** The Rot- $\pi$ DD representing  $\{526413, 526431, 562413, 562431\}$

of the  $k$ -prefix representing the vertex subset. Then, we obtain 526431, which represents the subgraph induced by  $\{2, 5\}$  and the reverse order of deletions.

By compressing swap sequences into a  $\pi$ DD, we can recursively construct a  $\pi$ DD for all topological orders. That is, for each recursion represented as a permutation  $\pi$ , if we apply  $\tau_{i,j}$  to delete  $\pi_i$ , we create the new  $\pi$ DD such that its root node is  $\tau_{i,j}$ , its 1-edge child is the  $\pi$ DD for swap sequences after applying  $\tau_{i,j}$ , and its 0-edge child is the  $\pi$ DD for swap sequences in which we do not apply  $\tau_{i,j}$ . The  $\pi$ DDs for the 1-edge and 0-edge child are recursively constructed.

However, deletions by swaps are not available for DP. In order to use DP approach, swap sequences for the same induced subgraph must be uniquely determined. Even if different prefixes of permutations represent the same induced subgraph, their swap sequences can differ. For example, consider the DAG in Fig. 1. Deletion sequences 314 and 134 generate the same induced subgraph on  $\{2, 5, 6\}$ , and these states are represented as 526413 and 625431, respectively. The induced subgraph on  $\{2, 5, 6\}$  has a topological order 526. In order to obtain this, we apply no swap to 526413, while we apply  $\tau_{1,3}$  to 625431. This means there are multiple  $\pi$ DDs corresponding to the same induced subgraph.

### 3.3 New Permutation Decision Diagrams: Rot- $\pi$ DDs

As described in the previous subsection, the DP approach cannot be used to directly construct a  $\pi$ DD. To overcome this problem, we use another decomposition where each vertex subset is uniquely represented as a prefix of permutations. In order to realize this, we use the left-rotation decomposition. A *left-rotation*  $\rho_{i,j}$  rearranges  $i$ th element into  $j$ th position, and  $k$ th element into  $(k-1)$ th position for each  $i+1 \leq k \leq j$ . That is,  $\rho_{i,j}$  rearranges an  $n$ -permutation  $\pi_1 \dots \pi_i \pi_{i+1} \dots \pi_j \dots \pi_n$  into  $\pi_1 \dots \pi_{i+1} \dots \pi_j \pi_i \dots \pi_n$ .

Left-rotations also can uniquely decompose a permutation. The left-rotation decomposition is similar to the one for swaps: we start with  $e^n$  and repeatedly apply  $\rho_{i,j}$  to move  $\pi_i$  to the  $j$ th position, from right to left. For example, consider to decompose 43152 into a sequence of left-rotations. We start with  $e^5 = 12345$ . Now, we move 2 from the 2nd position to the 5th position. Thus, we obtain  $13452 = \rho_{2,5}$ . Next, we move 5 from the 4th position to the 4th position, i.e., we do not rotate. Repeating this procedure, we finally obtain  $43152 = \rho_{2,5} \cdot \rho_{1,3} \cdot \rho_{1,2}$ .

---

**Algorithm 1.** Rot- $\pi$ DD construction for all topological orders of  $G = (V, E)$

---

```

ConstructRotPiDD( $G$ ):
if  $V$  is empty then
    return 1-sink
else if have never memorized the Rot- $\pi$ DD  $R_G$  for  $G$  then
    Rot-PiDD  $R \leftarrow$  0-sink
    for each  $v$  whose out-degree is 0 in  $G$  do
        Integer  $i \leftarrow v$ 's position in the increasing sequence of  $V$ ,  $j \leftarrow |V|$ 
         $R \leftarrow$  the Rot-PiDD with root node  $\rho_{i,j}$ , left child  $R$ , and right child
            ConstructRotPiDD( $G(V \setminus \{v\})$ )
    end for
    memorize  $R$  as  $R_G$ 
end if
return  $R_G$ 

```

---

Left-rotations realize the unique representation of an induced subgraph as a prefix of a permutation, because a prefix is always in an increasing order. Left-rotation  $\rho_{i,j}$  only changes the relative order between the  $i$ th element and the elements in  $[i + 1, j]$ , i.e., relative orders in  $[1, j - 1]$  are not changed. This means the  $(j - 1)$ -prefix is always in increasing order when we start with  $e^n$  and apply  $\rho_{i,j}$  in decreasing order of  $j$ .

Thus, we can use the DP approach by using left-rotations as node labels of  $\pi$ DDs. We call this left-rotation based  $\pi$ DD *Rot- $\pi$ DD*, and existing  $\pi$ DD *Swap- $\pi$ DD* to distinguish. Fig. 4 illustrates the Rot- $\pi$ DD for the same set as Fig. 2. Algorithm 1 describes the DP based construction algorithm of a Rot- $\pi$ DD.

### 3.4 Rot- $\pi$ DD Operations

Since Rot- $\pi$ DDs are decision diagrams, they can use the same set operations as Swap- $\pi$ DD such as union, intersection, and set difference. Some queries such as random samplings and counting the cardinality of the set represented by a Rot- $\pi$ DD are also available without any modification. The runtime of these operations depends on only the size of the Rot- $\pi$ DDs by using memo cache techniques.

On the other hand, some queries have to be redesigned. For example, the precedence query  $R.Precede(u, v)$  returns the Rot- $\pi$ DD that represents only permutations  $\pi$  extracted from the Rot- $\pi$ DD  $R$  such that  $u$  precedes  $v$  in  $\pi$ . This query is equivalent to addition of the edge  $(u, v)$  in a graph. This query can be designed as a recursive procedure described in Algorithm 2. The idea of the algorithm is simulation of moves of the two elements  $u$  and  $v$ . Initially, we start with the identity permutation, i.e.  $u$  and  $v$  are at the  $u$ th position and the  $v$ th position, respectively. After a rotation, the positions of  $u$  and  $v$  may be changed. If  $u$  or  $v$  are out of the range of later rotations, their relative order is fixed and we can check whether or not  $u$  precedes  $v$ . The runtime of a precedence query also depends on only the size of the Rot- $\pi$ DDs thanks to memo cache.

---

**Algorithm 2.** Precedence query for a Rot- $\pi$ DD  $R$ 

---

```

R.Precede( $u, v$ ):
if  $R$  is 0-sink then
  return 0-sink
else if  $R$  is 1-sink then
  if  $u < v$  then
    return 1-sink
  else
    return 0-sink
  end if
else
   $\rho_{x,y} \leftarrow$  the root node of  $R$ 
  if  $y < u$  and  $v < u$  then
    return 0-sink
  else if  $y < v$  and  $u < v$  then
    return  $R$ 
  else
     $R0 \leftarrow$  the left child of  $R$ ,  $R1 \leftarrow$  the right child of  $R$ 
    if  $x = v$  then
      return the Rot-PiDD with root node  $\rho_{x,y}$ , left child  $R0.Precede(u, v)$ , and
      right child  $R1$ 
    else if  $x = u$  then
      return  $R0.Precede(u, v)$ 
    else
      if  $x < u$  then
         $u \leftarrow u - 1$ 
      end if
      if  $x < v$  then
         $v \leftarrow v - 1$ 
      end if
      return the Rot-PiDD with root node  $\rho_{x,y}$ , left child  $R0.Precede(u, v)$ , and
      right child  $R1.Precede(u, v)$ 
    end if
  end if
end if
end if

```

---

## 4 Theoretical Analysis

In this section, we analyze the time and the space complexity of DP based counting and Rot- $\pi$ DD construction. Here, we remember the definition of  $IS(G)$ :  $IS(G)$  is the number of the induced subgraphs of  $G$  that can be obtained by deletions of vertices with out-degree 0. We start by proving the bound  $O(n^w)$  of  $IS(G)$ . According to Dilworth's theorem [7], the width  $w$  of a poset equals the *minimum path cover* of the DAG corresponding to the poset, where a path cover of a graph  $G$  is a set of paths in  $G$  such that each vertex of  $G$  must appear in at least one of the paths. Therefore, it is sufficient to prove the following theorem.

**Theorem 1.** *Given a DAG  $G$  with  $n$  vertices and minimum path cover  $w$ ,  $IS(G) \leq (n + 1)^w$  holds.*

*Proof.* Let  $p_i$  be the  $i$ th path of the minimum path cover and  $l_i$  be the length of  $p_i$ . Here, all vertices in a valid induced subgraph must be consecutive in prefix of each  $p_i$  due to precedence. The number of the possible prefixes of each path is at most  $l_i + 1$ , and the number of paths is  $w$ . Therefore,  $IS(G)$  is bounded by  $\prod_{k=1}^w (l_k + 1)$ . Since  $l_i$  is also bounded by  $n$ ,  $IS(G) \leq (n + 1)^w$  holds.  $\square$

In this proof, we use the rough estimation  $l_i = n$ , but in fact  $\sum_{k=1}^w l_k = n$  holds. We can prove a tighter bound using this restriction.

**Lemma 1.** *If  $\sum_{k=1}^w l_k = n$  holds,  $\prod_{k=1}^w (l_k + 1) \leq (n/w + 1)^w$  holds for all positive integers  $n$ ,  $1 \leq w \leq n$ , and  $1 \leq l_i \leq n$ .*

*Proof.* The proof can be done by induction. We omit details.  $\square$

**Corollary 1.** *Given a DAG  $G$  with  $n$  vertices and minimum path cover  $w$ ,  $IS(G) \leq (n/w + 1)^w$  holds.*

*Proof.* The proof follows from the proof of Theorem 1 and Lemma 1.  $\square$

Corollary 1 gives a new bound of  $IS(G)$ . Since  $(n/w + 1)^w$  is monotonically nondecreasing for all positive integers  $n$  and  $w$ , the range of  $(n/w + 1)^w$  is  $[n + 1, 2^n]$  for  $1 \leq w \leq n$ . This means the previous bound  $O(\min\{2^n, n^w\})$  can be directly replaced by  $O((n/w + 1)^w)$ . Hence, we obtain the time complexity  $O((n + m)(n/w + 1)^w)$  and the space complexity  $O((n/w + 1)^w)$  of the DP.

We can also estimate the size of a Rot- $\pi$ DD representing all topological orders and the time of the construction. The size of such a Rot- $\pi$ DD is at most  $w$  times larger than the space of DP because each DP recursion has at most  $w$  transitions, while each node of a Rot- $\pi$ DD has exactly two edges. Therefore, the size of such a Rot- $\pi$ DD is at most  $O(w(n/w + 1)^w)$ .<sup>1</sup> On the other hand, the time of the construction is as fast as DP, because each node is only created for each vertex deletion in constant time. Hence, the time complexity of the construction of a Rot- $\pi$ DD representing all topological orders is  $O((n + m)(n/w + 1)^w)$ .

## 5 Computational Experiments

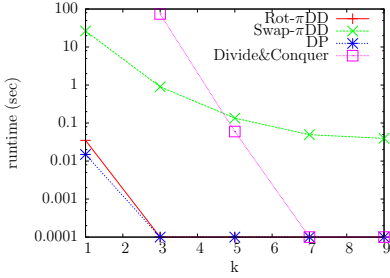
We measured the performance of our Rot- $\pi$ DD construction algorithm by computational experiments. Experiment setting is as follows.

- Input: A DAG.
- Output: The number of topological orders of the given DAG.
- Test Cases: For each  $n = 5, 10, 15, \dots, 45, 50$  and  $k = 1, 3, 5, 7, 9$ , we generate exactly 30 random DAGs with  $n$  vertices and  $\lfloor \frac{k}{10} \times \frac{n(n-1)}{2} \rfloor$  edges. (That is,  $k$  provides the edge density of DAGs.)

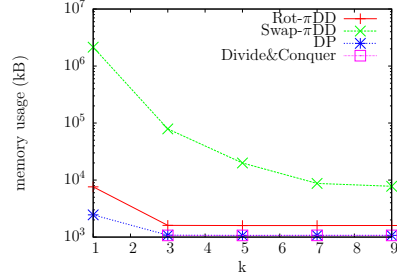
---

<sup>1</sup> Note that this bound is valid only for all topological orders. For any permutation set, the worst size of Rot- $\pi$ DDs is  $O(2^{n^2})$ , which is same as the size bound of Swap- $\pi$ DDs.





**Fig. 5.** Average runtime for construction when  $n = 20$



**Fig. 6.** Average memory usage for construction when  $n = 20$

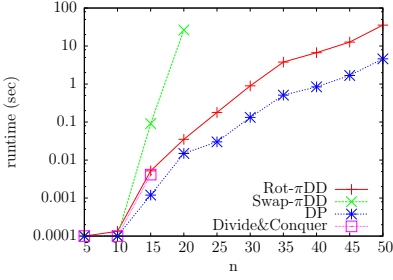
**Table 1.** Experimental results on the cases  $k = 1$

$n$	The number of topological orders	Rot- $\pi$ DD size	Time (sec)
5	60	16	0.00
10	270816	310	0.00
15	3849848730	3990	0.00
20	84248623806362	35551	0.04
25	1729821793136903967	179205	0.18
30	166022551499377802024339	695029	0.90
35	18897260805585874040859189398	2634015	3.78
40	192246224377065271125689349980187	4649639	6.68
45	7506858927008084384591070452622456252	8288752	12.69
50	375636607794991518114274279559952431497225	22542071	35.51

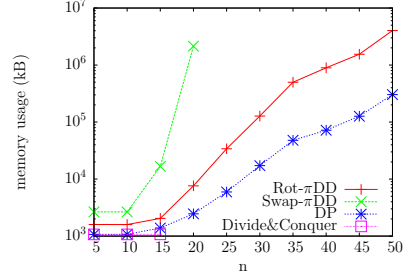
We also compared with other methods on the same setting. Comparisons are Swap- $\pi$ DD construction, DP counting, and Divide & Conquer counting [10]. Since direct construction of a Swap- $\pi$ DD is inefficient, we apply precedence queries for each edge individually. We implemented all algorithms in C++ and carried out experiments on a 3.20 GHz CPU machine with 64 GB memory.

Fig. 5 and Fig. 6 show the average runtime and memory usage on  $n = 20$  cases. Divide & Conquer method times-out on some cases of  $k = 1$ . These results indicate that the worse cases of all algorithms are sparse graphs. In general, sparse graphs tend to have a large poset width. In fact, the average  $w$  of  $k = 1$  cases is 10.6, while that of  $k = 5$  cases is 3.3. Therefore, the complexity  $O((n/w + 1)^w)$  also tends to become large on the sparse graph cases.

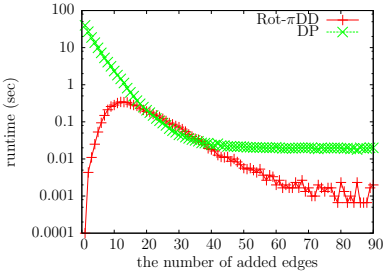
We therefore focus on sparse graphs. Table 1 shows the average numbers of topological orders, the sizes of Rot- $\pi$ DDs, and runtimes on the case  $k = 1$ . It shows the amazing efficiency of Rot- $\pi$ DDs:  $3.7 \times 10^{41}$  topological orders are compressed into a Rot- $\pi$ DD that has only  $2.2 \times 10^7$  nodes in 36 seconds on the case  $n = 50$ . Note that each node of Rot- $\pi$ DDs consumes about 30 bytes.



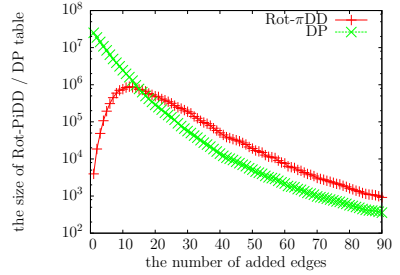
**Fig. 7.** Average runtime for construction when  $k = 1$



**Fig. 8.** Average memory usage for construction when  $k = 1$



**Fig. 9.** Average runtime for edge addition queries



**Fig. 10.** Average space for edge addition queries

Fig. 7 and Fig. 8 show the average runtime and memory usage on  $k = 1$  cases. Swap- $\pi$ DD and Divide & Conquer time-out on the case  $n \geq 25$  and  $n \geq 20$ , respectively. We can obtain a Rot- $\pi$ DD, which supports many operations for queries, with only tenfold increase in runtime and memory usage compared to DP. We guess that the overhead time is used to store new nodes of a Rot- $\pi$ DD into the hash table, and the overhead memory is caused by the difference of the space complexities between DP and Rot- $\pi$ DD as described in Section 4.

We also carried out experiments to measure the performance of query processing. On these experiments, we use 30 random DAGs with 25 vertices and 90 edges. We start with a graph having no edge, and add each edge individually. The Rot- $\pi$ DD method uses precedence queries for each edge addition, while DP recomputes  $TC(G)$  for each addition. We measure the runtime and the size of a Rot- $\pi$ DD and a DP table. Note that the DP table size equals  $IS(G)$ .

Fig. 9 and Fig. 10 show the results for query processing. In almost all cases, Rot- $\pi$ DDs can generate and index all topological orders faster than or equal to DP. Especially in sparse cases, query processing of Rot- $\pi$ DDs is very efficient. It may be because Rot- $\pi$ DDs (and Swap- $\pi$ DDs) can represent the set of all  $n$ -permutations with  $n(n-1)/2 + 1$  nodes (please refer to [8] for more details).

## 6 Conclusion

In this paper, we gave an efficient method for generating and indexing all topological orders of a given DAG. We proposed a new data structure Rot- $\pi$ DD, which is suitable for indexing topological orders. Theoretical analysis and experiments showed the efficiency of our construction algorithm, compression ratios of Rot- $\pi$ DDs, and query processing.

Future work is to apply Rot- $\pi$ DDs to solve several scheduling problems. We would like to develop new operations to process required queries and optimizations for each problem. Another topic is to apply the Rot- $\pi$ DD construction technique to other graph generation problems such as Hamiltonian paths and perfect elimination orderings. These problems can also be recursively divided into subproblems based on induced subgraphs.

## References

1. Atkinson, M.D.: On computing the number of linear extensions of a tree. *Order* **7**(1), 23–25 (1990)
2. Bender, M.A., Fineman, J.T., Gilbert, S.: A new approach to incremental topological ordering. In: 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1108–1115. Society for Industrial and Applied Mathematics (2009)
3. Brightwell, G., Winkler, P.: Counting linear extensions. *Order* **8**(3), 225–242 (1991)
4. Bublely, R., Dyer, M.: Faster random generation of linear extensions. *Discrete Mathematics* **201**(1), 81–88 (1999)
5. Cooper, J.N.: When is linear extensions counting easy? AMS Southeastern Sectional Meeting (2013)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*. The MIT Press, Cambridge (2001)
7. Dilworth, R.P.: A decomposition theorem for partially ordered sets. *Annals of Mathematics* **51**(1), 161–166 (1950)
8. Inoue, Y.: Master’s thesis: Generating PiDDs for indexing permutation classes with given permutation patterns. Tech. Rep. TCS-TR-B-14-9, Division of Computer Science, Hokkaido University (2014)
9. Kahn, A.B.: Topological sorting of large networks. *Communications of the ACM* **5**(11), 558–562 (1962)
10. Li, W.N., Xiao, Z., Beavers, G.: On computing the number of topological orderings of a directed acyclic graph. *Congressus Numerantium* **174**, 143–159 (2005)
11. Minato, S.:  $\pi$ DD: A new decision diagram for efficient problem solving in permutation space. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 90–104. Springer, Heidelberg (2011)
12. Ono, A., Nakano, S.: Constant time generation of linear extensions. In: Liśkiewicz, M., Reischuk, R. (eds.) FCT 2005. LNCS, vol. 3623, pp. 445–453. Springer, Heidelberg (2005)
13. Pearce, D.J., Kelly, P.H.: A dynamic topological sort algorithm for directed acyclic graphs. *ACM Journal of Experimental Algorithmics* **11**(1.7), 1–24 (2006)
14. Pruesse, G., Ruskey, F.: Generating linear extensions fast. *SIAM Journal on Computing* **23**(2), 373–386 (1994)
15. Tarjan, R.E.: Depth-first search and linear graph algorithms. *SIAM Journal on Computing* **1**(2), 146–160 (1972)

# **Matching and Assignment I**

# Planar Matchings for Weighted Straight Skeletons

Therese Biedl<sup>1</sup>, Stefan Huber<sup>2</sup>(✉), and Peter Palfrader<sup>3</sup>

<sup>1</sup> David R. Cheriton School of Computer Science, University of Waterloo,  
Waterloo, ON N2L 1A2, Canada

biedl@uwaterloo.ca

<sup>2</sup> Institute of Science and Technology Austria, 3400 Klosterneuburg, Austria  
stefan.huber@ist.ac.at

<sup>3</sup> FB Computerwissenschaften, Universität Salzburg, 5020 Salzburg, Austria  
palfrader@cosy.sbg.ac.at

**Abstract.** In this paper, we introduce planar matchings on directed pseudo-line arrangements, which yield a planar set of pseudo-line segments such that only matching-partners are adjacent. By translating the planar matching problem into a corresponding stable roommates problem we show that such matchings always exist.

Using our new framework, we establish, for the first time, a complete, rigorous definition of weighted straight skeletons, which are based on a so-called wavefront propagation process. We present a generalized and unified approach to treat structural changes in the wavefront that focuses on the restoration of weak planarity by finding planar matchings.

**Keywords:** Planar matchings · Pseudo-line arrangements · Stable roommates · Weighted straight skeletons

## 1 Introduction

The straight skeleton is a skeletal structure of a polygon  $P$ , similar to the Voronoi diagram. It was introduced to computational geometry almost two decades ago by Aichholzer et al. [1], and its definition is based on a so-called wavefront propagation process, see Fig. 1: Each edge of  $P$  emits a wavefront edge that moves towards the interior of  $P$  at unit speed in a self-parallel manner. The polygons formed by these wavefront edges at any given time  $t \geq 0$  are the wavefront, denoted by  $\mathcal{W}_P(t)$ , and take the form of a mitered offset of  $P$ . Over time, the wavefront undergoes two different kinds of topological changes, so-called events, due to self-interference: roughly speaking, an *edge event* happens when a wavefront edge collapses, and a *split event* happens when the wavefront splits into parts. The straight skeleton  $\mathcal{S}(P)$  of  $P$  is then defined as the geometric graph whose edges are the traces of the vertices of  $\mathcal{W}_P$ . Similar to Voronoi

---

T. Biedl was supported by NSERC and the Ross and Muriel Cheriton Fellowship.

P. Palfrader was supported by Austrian Science Fund (FWF): P25816-N15.

diagrams and the medial axis, straight skeletons became a versatile tool for applications in various domains of science and industry [8].

The *weighted straight skeleton*, where wavefront edges do not necessarily move at unit speed, was first mentioned by Eppstein and Erickson [5] and has since been used in a variety of different applications [2, 7, 9, 10]. Weighted straight skeletons, with both positive and negative weights, also constitute a theoretical tool to generalize straight skeletons to 3D [3]. Even though weighted straight skeletons have already been applied in both theory and practice, only recently Biedl et al. [4] showed that basic properties of unweighted straight skeletons do not carry over to weighted straight skeletons in general. Biedl et al. [4] also proposed solutions for an ambiguity in the definition of straight skeletons caused by certain edge events and first mentioned by Kelly and Wonka [9] and Huber [8].

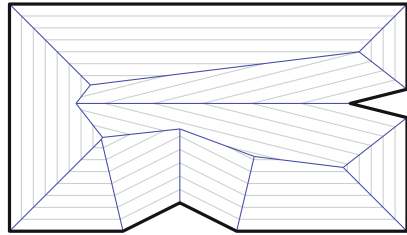
In this paper, we discuss another open problem in the definition of weighted straight skeletons caused by split events. An event happens due to a topological change in the wavefront and the event handling was so far guided by one fundamental principle: *Between events, the wavefront is a planar collection of wavefront polygons*. This is easily achieved when handling edge events and “simple” split events. However, is it always possible to handle multiple simultaneous, co-located split events in a fashion that respects this fundamental principle?

We will show that it is necessary to weaken the requirement of strict planarity in the fundamental principle. After that, we can answer the question to the affirmative and therefore show how to define weighted straight skeletons safely in the presence of multiple simultaneous, co-located split events. (Note that due to the discontinuous character of straight skeletons, it is not possible to tackle this problem by means of simulation of simplicity.) We first rephrase this problem as a *planar matching* problem of directed pseudo-lines and show how to transform the planar matching problem into a *stable roommate* problem. For the main result, we prove that our particular stable roommate problem always possesses a solution and those solutions tell us how to do the event handling of the wavefront in order to maintain planarity.

## 2 Weighted Straight Skeletons

### 2.1 The Wavefront

Let  $P$  denote a polygon, possibly with holes. We denote by  $\sigma(e) \in \mathbb{R} \setminus \{0\}$  the *weight* of the edge  $e$  of  $P$  and call  $\sigma$  the *weight function*. For every edge  $e$  of  $P$ , let  $n(e)$  denote the normal vector of  $e$  that points to the interior of  $P$ . Initially,



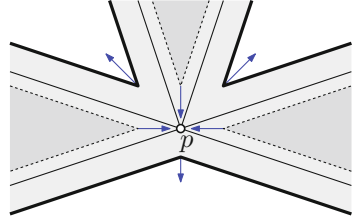
**Fig. 1.** The straight skeleton  $\mathcal{S}(P)$  (blue) of a polygon  $P$  (bold) is defined as the traces of wavefront vertices over time. Instances of the wavefront  $\mathcal{W}_P(t)$  at different times  $t$  are shown in gray.

every edge of  $P$  sends out a wavefront edge with fixed speed  $\sigma(e)$ . That is, the segments of the wavefront  $\mathcal{W}(t)$  at time  $t$  that originate from edge  $e$  are contained in  $\bar{e} + t \cdot \sigma(e) \cdot n(e)$ , where  $\bar{e}$  denotes the supporting line of  $e$ . If  $\sigma(e)$  is negative, the wavefront edge that emanated from  $e$  moves to the exterior of  $P$ .

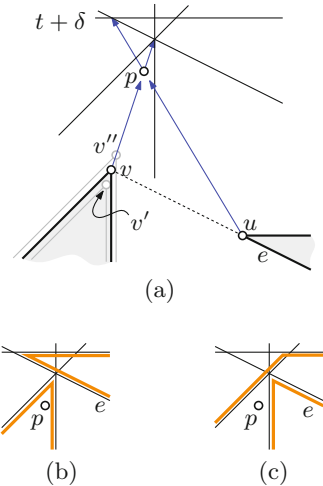
Intuitively, an event happens when a wavefront vertex meets another wavefront edge or, in particular, another wavefront vertex. The situation becomes more complicated when two or more such events are co-located at the same time  $t$ . For unweighted straight skeletons, i.e., with all weights set to 1, the wavefront is planar between events, and we can interpret events as the incidences where planarity is violated. Let us consider the case where multiple wavefront vertices meet at a point  $p$ . For ordinary straight skeletons, we restore planarity by considering the cyclical order of wavefront edges meeting at  $p$  and by re-pairing each edge with a cyclically neighboring edge, see Fig. 2. We call this the *standard pairing technique*.

In case of weighted straight skeletons, the situation becomes significantly more difficult. First, (strict) planarity cannot always be restored. Second, (weak) planarity may not be restored by the simple pairing scheme mentioned above, and it is not even obvious why some other pairing scheme that restores planarity must exist.

Consider Fig. 3, where two vertices,  $u$  and  $v$ , meet simultaneously at point  $p$  and time  $t$ . By construction, the vertex  $v$  lies on the supporting line of  $e$  for a positive-length time interval. We show the supporting lines of the edges at time  $t + \delta$ , with  $\delta$  being positive but small. We have three combinatorial possibilities to pair up the wavefront edges. One of them leads to a crossing. The other two possibilities are illustrated in Fig. 3 (b, c). Both remaining possibilities are not planar in a strict sense. Still, there are no crossings—instead edges only



**Fig. 2.** A wavefront before (dotted), at (solid), and after an event (bold), with blue arrows showing movement direction of wavefront vertices. The *standard pairing technique* for handling a split event pairs each edge with its other neighbor in the cyclical order.



**Fig. 3.** Two wavefront vertices meet at  $p$ . There are two possibilities, (b) and (c), in order to pair up the edges such that the wavefront remains planar in a weak sense.

touch. This shows there is no way to pair up the edges and remain strictly-planar.

Let us suppose that we initially move  $v$  slightly away from  $p$  or slightly closer to  $p$ . We obtain  $v'$  and  $v''$  respectively, see Fig. 3 (a). We adapt their speeds such that they still reach  $p$  at time  $t$ . Since  $v'$  moves slightly faster than  $v$  and  $e$ , at time  $t$ , the vertex  $v'$  overtakes  $e$ . Similarly,  $e$  overtakes  $v''$ . Hence, if we replace  $v$  by  $v'$ , the pairing in Fig. 3 (b) becomes invalid, and if we replace  $v$  by  $v''$ , the pairing in Fig. 3 (c) becomes invalid. In particular, for the latter case the only valid pairing is the original pairing and the standard pairing technique fails.

For our further discussions it will be necessary to define precisely what we mean by *event* or *weak planarity*. Let  $\Phi$  denote the set of all straight-line embeddings  $\varphi: V \rightarrow \mathbb{R}^2$  of a graph  $G = (V, E)$ . The pair  $(\Phi, \|\cdot\|_\infty)$  constitutes a normed space, where  $\|\cdot\|_\infty$  is defined by  $\|\varphi\|_\infty = \max_{v \in V} \|\varphi(v)\|$ . Note that the set of planar<sup>1</sup> straight-line embeddings is an open subset of  $\Phi$  w.r.t. the usual topology induced by  $\|\cdot\|_\infty$ .

**Definition 1.** *The set of weakly-planar embeddings of  $G$  is the topological closure of the set of planar embeddings of  $G$ .*

This implies that every planar embedding is weakly-planar as well. In addition, for every weakly-planar embedding  $\varphi$  and for every  $\varepsilon > 0$  there is a planar  $\varepsilon$ -perturbation  $\varphi'$  of  $\varphi$ , that is,  $\|\varphi - \varphi'\|_\infty < \varepsilon$ . This definition allows us now to rephrase the fundamental principle as follows:

*At all times, the wavefront is a weakly-planar collection of polygons.*

**Events.** The wavefront  $\mathcal{W}$  is initially weakly-planar. Informally, an event occurs when the wavefront is about to cease being weakly-planar and event handling needs to restructure the wavefront locally such that it can continue propagating in a weakly-planar fashion.

Assume that  $\mathcal{W}(t')$  remains weakly-planar for all  $t' \in [t - \delta, t]$  and some  $\delta > 0$ . For this time interval, we can consider  $\mathcal{W}$  to be a kinetic planar straight-line graph with a fixed set of kinetic vertices and edges. For Definition 2, we fix the vertex and edge set of  $\mathcal{W}$ , including the velocities of the vertices and temporarily ignore event handling. Furthermore, we denote by  $B(p, r)$  the closed disk centered at  $p$  with radius  $r$  and by  $\mathcal{W}(t') \cap B(p, r)$  the planar straight-line graph  $\mathcal{W}(t')$  with all edges truncated to fit into  $B(p, r)$  or removed if they entirely reside outside  $B(p, r)$ .

**Definition 2.** *At location  $p$  and time  $t$  an event happens if at least two vertices meet at time  $t$  at  $p$  or if  $\exists \varepsilon_0 > 0 \forall \varepsilon \in (0, \varepsilon_0) \exists \delta > 0$  such that*

- (i)  $\mathcal{W}(t') \cap B(p, \varepsilon)$  is non-empty and weakly-planar for  $t' \in [t - \delta, t]$  and
- (ii)  $\mathcal{W}(t') \cap B(p, \varepsilon)$  is non-empty and not weakly-planar for  $t' \in (t, t + \delta]$ .

*We call the edges that meet  $p$  at time  $t$  the edges which are involved in the event.*

<sup>1</sup> A straight-line embedding  $\varphi$  is called *planar* if its edges do not intersect except at common endpoints.



As this definition defines events localized at some point  $p$ , we can also talk about multiple events occurring at the same time  $t$  at different locations. If an event happens at location  $p$  and time  $t$  then, typically, weak planarity of  $\mathcal{W}$  is violated locally around  $p$  after time  $t$ . However, weak planarity is not violated if, for instance, a wavefront polygon collapses to a point. Fig. 3 gives another example where weak planarity is not violated after the event. The goal of event handling is to restore weak planarity by locally adapting the wavefront structure. We also want to remark that in certain cases multiple ways to correctly handle an event may exist, where one solution yields only a weakly-planar wavefronts while a different one produces a strictly-planar wavefront after the event.

**Definition 3.** *We call the event at location  $p$  and time  $t$  elementary if three edges are involved. We call it an edge event if  $B(p, \varepsilon) \setminus \mathcal{W}(t - \delta)$  consists of two connected components and a split event otherwise. Non-elementary edge and split events are called multi-edge and multi-split events respectively.*

It is known how to handle edge events and elementary split events [4]. In the following, we present one unified approach that is able to correctly handle any type of event, including, in particular, multi-split events. Consequently, one side effect of our definition of weighted straight skeletons is that the distinction between edge events and split events becomes unnecessary.

## 2.2 Pairing Edges

Assume an event happens at time  $t$  at location  $p$ . Up until time  $t$  the wavefront  $\mathcal{W}$  is weakly-planar, and it becomes not weakly-planar after  $t$ . In order to restore weak planarity, we have to transform the wavefront structure. This involves repairing of wavefront edges.

We reduce the problem of pairing up wavefront edges during event handling to a particular matching problem, discussed in Section 3. This problem, which we study independently of straight skeletons, takes a pseudo-line arrangement in general position as input and provides us with a means to construct a weakly-planar wavefront again. In the following, we describe how to transform a weakly-planar wavefront into a suitable pseudo-line arrangement for the matching problem.

The pseudo-lines stem from the supporting lines of wavefront edges and are required to be in general position. By *general position* we mean that any pair of lines intersect in exactly one unique point. In particular, this implies that no two lines are parallel, no two lines are identical, and no three lines intersect in a common point.

At time  $t$ , several edges of the wavefront  $\mathcal{W}$  are incident at location  $p$ . For each such edge, either zero, one, or both endpoints approach  $p$  at time  $t$ . We construct a simplified version of the wavefront, denoted by  $\mathcal{W}'$ , by dropping edges where both endpoints reach  $p$  and joining its two endpoints. Furthermore, any edge where no endpoint reaches  $p$  is split into two edges by a new wavefront vertex that also reaches  $p$  at time  $t$ . Thus, in  $\mathcal{W}'$  an even number of wavefront edges have exactly one endpoint at point  $p$  at time  $t$ , see Fig. 4.

Next, we choose  $\varepsilon$  and  $\delta$  sufficiently small, such that no other event happens between  $t$  and  $t + \delta$  and that exactly the edges involved in the event intersect  $B(p, \varepsilon)$  during the interval  $[t, t + \delta]$ . We obtain  $\mathcal{W}''$  from  $\mathcal{W}'$  by perturbing the locations of its vertices. This perturbation shall satisfy the following properties: (i) The edges involved in the event still reach  $p$  at time  $t$ . (ii) The supporting lines of involved edges are in general position at time  $t + \delta$ . (iii) The perturbation is such that  $\mathcal{W}''$  is strictly-planar everywhere outside  $B(p, \varepsilon)$  at time  $t + \delta$ . (iv) The perturbation is such that any vertex is (at time  $t + \delta$ ) on the same side of the supporting line through any edge in both  $\mathcal{W}'$  and  $\mathcal{W}''$ . The set of supporting lines at time  $t + \delta$  then shall be the input to the matching problem.

We use the new pairing obtained from the matching algorithm to construct a new (still perturbed) wavefront  $\mathcal{W}'''$  from  $\mathcal{W}''$ . The new pairing ensures that  $\mathcal{W}'''$  is strictly-planar around  $p$  after time  $t$ , see Lemma 7 in Section 3.3. If several multi-split events happen at the same time, then this procedure is repeated for every such event independently. Each event will locally restore strict planarity, and, thus, global strict planarity will be restored. Finally, we revert the perturbation on  $\mathcal{W}'''$  and obtain the new post-event wavefront.

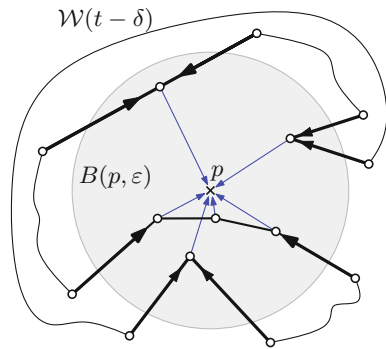
**Lemma 1.** *The new post-event wavefront  $\mathcal{W}^*$  is weakly-planar.*

*Proof.* Note that the perturbation we apply to obtain  $\mathcal{W}''$  from  $\mathcal{W}'$  was sufficiently small such that no vertex could “jump” over the supporting line of any edge of the wavefront. Therefore, if we assume to the contrary that  $\mathcal{W}^*$  is not weakly-planar, that would imply that the perturbed wavefront  $\mathcal{W}'''$  was not (strictly) planar either. Since  $\mathcal{W}'''$  is (strictly) planar outside of  $B(p, \varepsilon)$  per our requirement for the perturbation and is (strictly) planar within  $B(p, \varepsilon)$  due to the new pairing, this is a contradiction.

### 3 Matchings and Roommates

For an even  $N$ , let  $\mathcal{L} = \{\ell_1, \dots, \ell_N\}$  be an oriented pseudo-line arrangement in general position, i.e., a set of directed Jordan-curves that begin and end at infinity and intersect each other in single, unique points. Let  $\mathcal{C}$  be a pseudo-circle that encloses all intersections of pseudo-lines and that intersects each (directed) pseudo-line  $\ell$  exactly twice, once in its *begin-point*  $b(\ell)$  and once in its *end-point*.

A *matching*  $M$  in  $\mathcal{L}$  is a grouping of  $\ell_1, \dots, \ell_N$  into pairs. The *matching tail* of  $\ell_i$  is the sub-curve of  $\ell_i$  from  $b(\ell_i)$  to  $\ell_i \times M(\ell_i)$ , i.e., the point where  $\ell_i$  intersects its matching-partner  $M(\ell_i)$ .



**Fig. 4.** A multi-split event occurs at location  $p$ . The involved edges form 4 chains. The simplified wavefront  $\mathcal{W}'$  contains exactly 8 edges, shown in bold, stemming from those chains.

**Definition 4.** A matching in  $\mathcal{L}$  is called planar if the union of the matching tails gives a planar drawing.

The *planar matching problem* is the problem of finding a planar matching  $M$  for a given pseudo-line arrangement  $\mathcal{L}$  in general position. In the following we translate the planar matching problem into a stable roommate problem.

### 3.1 The Stable Roommate Problem

Assume that we have an even number  $N$  of elements  $\mathcal{A} = \{a_1, \dots, a_N\}$ . Each element has a ranking of elements, which is *complete* and *strict*, i.e., all elements are ranked and no two elements are ranked the same. Let  $M$  be a matching of  $a_1, \dots, a_N$ . A pair  $\{a_i, a_j\}$  is a *blocking pair* for  $M$  if  $a_i$  prefers  $a_j$  over  $M(a_i)$  and  $a_j$  prefers  $a_i$  over  $M(a_j)$ . A matching is *stable* if there is no blocking pair. The *stable roommate problem* asks for a stable matching in  $\mathcal{A}$ . The stable roommate problem is a well-studied problem in optimization theory (see, for example, Fleiner et al. [6] and the references therein). In particular, not every instance of the stable roommate problem has a solution, and testing whether it has a solution can be done in polynomial time.

Let us again consider the directed pseudo-line arrangement  $\mathcal{L}$ . As we walk along a pseudo-line  $\ell_i$  from its begin-point to its end-point we encounter all other pseudo-lines in  $\mathcal{L}$ . This order naturally gives us a complete and strict ranking for  $\ell_i$  if we attach  $\ell_i$  itself at the end of the list. Thus,  $\mathcal{L}$  defines an instance of the stable roommate problem.

**Lemma 2.** A directed pseudo-line arrangement has a planar matching if and only if the corresponding stable roommate instance has a stable matching.

*Proof.* For a matching  $M$ , the matching tails of two pseudo-lines  $\ell_i, \ell_j$  cross if and only if  $\ell_i$  prefers  $\ell_j$  over  $M(\ell_i)$  and  $\ell_j$  prefers  $\ell_i$  over  $M(\ell_j)$ . Hence, the matching is non-planar if and only if there is a blocking pair.

### 3.2 Stable Partitions

In order to solve our particular stable roommate problem, we review some results on so-called stable partitions, mostly based on a paper by Tan and Hsueh [12].

Let  $\mathcal{A}$  be an instance of a stable roommate problem, and let  $\pi$  be a permutation on  $\mathcal{A}$ , i.e., a bijective map  $\mathcal{A} \rightarrow \mathcal{A}$ . This function partitions  $\mathcal{A}$  into one or more *cycles*, i.e., sequences  $a'_0, \dots, a'_{k-1}$  in  $\mathcal{A}$  with  $a'_0 \xrightarrow{\pi} a'_1 \xrightarrow{\pi} \dots \xrightarrow{\pi} a'_{k-1} \xrightarrow{\pi} a'_0$ . A cycle with  $k \geq 3$  is called a *semi-party cycle* if  $a'_i$  prefers  $\pi(a'_i)$  over  $\pi^{-1}(a'_i)$ . A *semi-party partition* of  $\mathcal{A}$  is a permutation of  $\mathcal{A}$  where all cycles with  $k \geq 3$  are semi-party cycles.

Given a semi-party partition  $\pi$ , a pair  $\{a_i, a_j\}$  is called a *party-blocking pair* if  $a_i$  prefers  $a_j$  over  $\pi^{-1}(a_i)$  and  $a_j$  prefers  $a_i$  over  $\pi^{-1}(a_j)$ . A *stable partition* is a semi-party partition that has no party-blocking pairs. The cycles of a stable partition are called *parties*. An odd (even) party is a party of odd (even) cardinality. Furthermore,  $a_i, a_j$  are *party-partners* if  $a_i = \pi(a_j)$  or  $a_j = \pi(a_i)$ .

**Theorem 1** ([11, 12]). *For any instance  $\mathcal{A}$  of the stable roommate problem the following statements hold:*

1.  $\mathcal{A}$  has a stable partition, and it can be found in polynomial time.
2. Any stable partition of  $\mathcal{A}$  has the same number of odd parties.
3.  $\mathcal{A}$  has a stable matching if and only if it has a stable partition with no odd parties.

### 3.3 Existence of Planar Matchings

Now we consider parties that occur in stable roommate instances defined by a directed pseudo-line arrangement  $\mathcal{L}$ . [Theorem 1](#)(1) gives us a stable partition  $\pi$  for  $\mathcal{L}$ . Let a *singleton-party*, a *pair-party*, and a *cycle-party* be a party consisting of one, two, and at least three pseudo-lines, respectively. For all pseudo-lines  $\ell$  that are not a singleton-party, let their *party-tail* be the part between  $b(\ell)$  and  $\ell \times \pi^{-1}(\ell)$ . For any pseudo-line  $\ell$  that is a singleton-party, let its *party-tail* be the part of  $\ell$  between begin-point and end-point.

**Lemma 3.** *The party-tails of two pseudo-lines  $\ell, \ell'$  do not intersect unless  $\ell$  and  $\ell'$  are party-partners.*

*Proof.* Assume that  $\ell \times \ell'$  belongs to both party-tails, but  $\ell$  and  $\ell'$  are not party-partners. We first show that  $\ell$  prefers  $\ell'$  over  $\pi^{-1}(\ell)$ . This holds automatically if  $\ell$  is a singleton-party, because then  $\pi^{-1}(\ell) = \ell$ , and any pseudo-line ranks itself lowest. If  $\ell$  is not a singleton-party, then the party-tail of  $\ell$  consists of the sub-curve between  $b(\ell)$  and  $\ell \times \pi^{-1}(\ell)$ . Since  $\ell' \neq \pi^{-1}(\ell)$  by assumption, and since no three pseudo-lines intersect in a point,  $\ell \times \ell'$  comes strictly earlier than  $\ell \times \pi^{-1}(\ell)$  when walking along  $\ell$ . By definition of the ranking for directed pseudo-lines, hence  $\ell$  prefers  $\ell'$  over  $\pi^{-1}(\ell)$ .

Similarly one shows that  $\ell'$  prefers  $\ell$  over  $\pi^{-1}(\ell')$ . Hence,  $\{\ell, \ell'\}$  is a party-blocking pair and  $\pi$  is not a stable partition, a contradiction.

**Lemma 4.** *There cannot be two singleton-parties.*

*Proof.* Assume that  $P$  and  $P'$  are two singleton-parties, with  $P = \{\ell\}$  and  $P' = \{\ell'\}$ . Since they are singleton-parties, their party-tails extend from their begin-points to their end-points. Since all pseudo-lines intersect within  $\mathcal{C}$ , so do  $\ell$  and  $\ell'$ . But  $\ell$  and  $\ell'$  are not party-partners, in contradiction to [Lemma 3](#).

**Lemma 5.** *There cannot be two cycle-parties.*

*Proof.* Assume we have two cycle-parties  $P_1 = \{\ell_0, \ell_1, \dots, \ell_{a-1}\}$  and  $P_2 = \{\ell'_0, \ell'_1, \dots, \ell'_{b-1}\}$ , with  $\pi(\ell_i) = \ell_{i+1}$ , addition modulo  $a$ , and  $\pi(\ell'_i) = \ell'_{i+1}$  with addition modulo  $b$ .

Let  $G(P_1)$  be the graph formed by the party tails of  $P_1$  as follows: The vertex set comprises  $b(\ell)$  and  $\ell \times \pi(\ell)$  for every pseudo-line  $\ell$  in  $P_1$ . We add each party-tail as two edges  $(b(\ell), \ell \times \pi(\ell))$  and  $(\ell \times \pi(\ell), \pi^{-1}(\ell) \times \ell)$ , see [Fig. 5](#).

Note that  $G(P_1)$  has the following structure: It consists of a cycle  $\mathcal{C}_1$  of edges of the form  $(\ell \times \pi(\ell), \pi^{-1}(\ell) \times \ell)$  together with one edge attached to each vertex of  $\mathcal{C}_1$  of the form  $(b(\ell), \ell \times \pi(\ell))$ . By Lemma 3,  $G(P_1)$  is planar. Note that the vertices  $b(\ell_0), b(\ell_1), \dots, b(\ell_{a-1})$  lie on  $\mathcal{C}$  and are ordered clockwise or counterclockwise. Therefore,  $G(P_1)$  tessellates the area enclosed by  $\mathcal{C}$  into  $a + 1$  regions. Note that exactly  $a$  of those regions are partially bounded by  $\mathcal{C}$ . The remaining region is the one bounded by  $\mathcal{C}_1$ . Similarly, we define  $G(P_2)$  and  $\mathcal{C}_2$ .

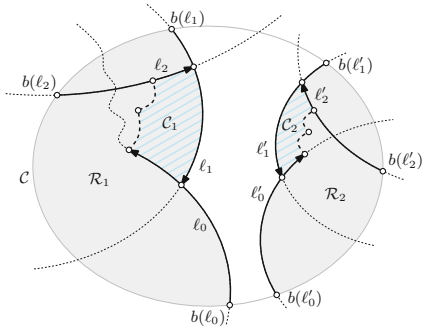


Fig. 5. The two pseudo-lines  $\ell_0$  and  $\ell'_0$  cannot intersect

Again by Lemma 3,  $G(P_1) \cup G(P_2)$  is planar, and it follows that  $G(P_2)$  is entirely contained in one region of  $G(P_1)$ . This region is not the region bounded by  $\mathcal{C}_1$ . We denote by  $\mathcal{R}_1$  the union of all regions of  $G(P_1)$  that do not contain  $G(P_2)$ . Likewise, we denote by  $\mathcal{R}_2$  the union of all regions of  $G(P_2)$  that do not contain  $G(P_1)$ . We observe that  $\mathcal{R}_1$  and  $\mathcal{R}_2$  are disjoint.

Without loss of generality, the boundary of  $\mathcal{R}_1$  consists of parts of  $\mathcal{C}$  as well as the path  $b(\ell_1), \ell_1 \times \ell_2, \ell_0 \times \ell_1, b(\ell_0)$ . Likewise,  $\mathcal{R}_2$  is bounded by parts of  $\mathcal{C}$  and edges stemming from  $\ell'_1$  and  $\ell'_0$ .

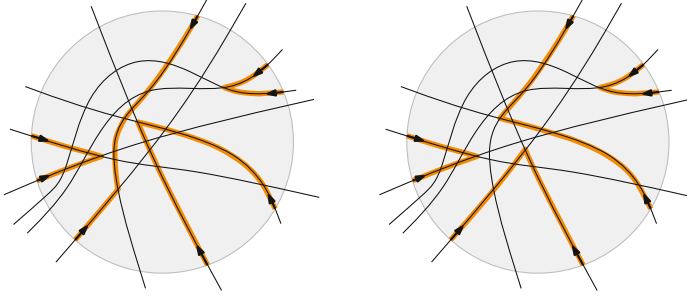
In the following, we will show that  $\ell_0$  cannot intersect  $\mathcal{R}_2$ , and, conversely,  $\ell'_0$  cannot intersect  $\mathcal{R}_1$ . Consequently,  $\ell_0$  does not intersect  $\ell'_0$  within  $\mathcal{C}$ , which is a contraction as we require each pair of pseudo-lines to intersect exactly once in the area enclosed by  $\mathcal{C}$ . This concludes the proof.

Note that  $\ell_0$  starts at  $b(\ell_0)$ , then makes up parts of the boundary of  $\mathcal{R}_1$  until it reaches  $\ell_0 \times \ell_1$ . Then,  $\ell_0$  moves into the interior of  $\mathcal{R}_1$  as it makes up an edge of  $\mathcal{C}_1$ , but not the one that is part of the boundary of  $\mathcal{R}_1$ . Once  $\ell_0$  enters  $\mathcal{R}_1$ , it can leave only by intersecting  $\mathcal{C}$  at its end-point as it is not allowed to self-intersect or to intersect  $\ell_1$  a second time. After  $\ell_0$  has left the area enclosed by  $\mathcal{C}$ , it cannot enter again, as it intersects  $\mathcal{C}$  exactly twice. Likewise,  $\ell'_0$  will exit the area enclosed by  $\mathcal{C}$  through its end-point in  $\mathcal{R}_2$  and cannot intersect  $\mathcal{R}_1$ .

**Lemma 6.** *There cannot be a singleton-party and a cycle-party.*

*Proof.* We follow the same idea as in the previous proof, and use  $P_1$  as the cycle-party and  $P_2$  as the singleton-party. Let  $\ell_0$  be defined as previously, and use  $\ell'_0$  as the single line in  $P_2$ . Since the tail of  $\ell'_0$  consists of all points between the begin-point and the end-point of  $\ell'_0$ , again no intersection between  $\ell_0$  and  $\ell'_0$  is possible.

**Theorem 2.** *No instance of a stable roommate problem defined by a directed pseudo-line arrangement  $\mathcal{L}$  can have an odd party.*



**Fig. 6.** A planar matching of pseudo-lines specifies how to construct a weakly-planar post-event wavefront. One arrangement may have multiple planar matchings.

*Proof.* Assume to the contrary that some stable partition  $\pi$  has an odd party  $P$ . As  $\mathcal{L}$  comprises an even number of pseudo-lines, there needs to be another odd party  $P'$ . This can only happen if there are two singleton-parties, two (odd) cycle parties, or a singleton-party and an (odd) cycle-party. These are ruled out by [Lemma 4](#), [Lemma 5](#), and [Lemma 6](#), respectively.

**Theorem 3.** *Every directed pseudo-line arrangement has a planar matching, and it can be found in polynomial time.*

*Proof.* This is a direct result of [Lemma 2](#), [Theorem 1](#), and [Theorem 2](#).

By [Theorem 1](#) we can find a stable partition in polynomial time. By [Theorem 2](#) and [Lemma 5](#), it consists of pair-parties, except for at most one cycle-party  $P$  that has even length. If there is no cycle-party, then the stable partition is in fact a stable matching. Otherwise, if say  $\ell_1, \dots, \ell_N$  is the even cycle-party, then we can find stable matching  $M$  easily, and there are two choices: Either set  $M(\ell_{2i}) = \pi(\ell_{2i})$  and  $M(\ell_{2i+1}) = \pi^{-1}(\ell_{2i+1})$ , or do the same after shifting all indices by one.

### 3.4 Application to Straight Skeletons

The matching tails of the pseudo-lines play the role of wavefront edges after the event. The matching tells us how to pair up the wavefront edges in order to restore planarity locally at  $p$ , see [Fig. 6](#).

**Lemma 7.** *There exists a weakly-planar wavefront after the event if there is a planar matching for  $\mathcal{L}$ .*

Using [Theorem 3](#), we have found a stable matching and with it a weakly-planar post-event wavefront, in polynomial time. Notice that if a cycle-party exists, then there are two possible post-event wavefronts. Consequently, ambiguities in the development of the wavefront may be caused by edge events between parallel edges [\[4\]](#) and multi-split events alike.

## 4 Conclusion

Although algorithms and even rudimentary implementations to construct the weighted straight skeleton were previously presented, and even though several applications are suggested in the literature, this paper is the first to provide a concrete, constructive proof that a well-defined weighted straight skeleton actually exists in all cases. This result is based on two main ingredients: First, we introduced and studied planar matchings on a directed pseudo-line arrangement as a generic tool independent of straight skeletons. In particular, we showed that planar matchings always exist. Second, our interpretation of an event as violation of (weak) planarity unifies the classification of edge and split events in 2D and promises to simplify the description and study of straight skeletons in dimensions higher than two, where the number of types and complexity of events would significantly increase otherwise.

**Acknowledgments.** We would like to thank David Eppstein for the idea of interpreting the edge-pairing problem as a stable roommate problem.

## References

1. Aichholzer, O., Alberts, D., Aurenhammer, F., Gärtner, B.: Straight Skeletons of Simple Polygons. In: Proc. 4th Int'l Symp. of LIESMARS, pp. 114–124 (1995)
2. Aurenhammer, F.: Weighted Skeletons and Fixed-Share Decomposition. *Comput. Geom. Theory and Appl.* **40**(2), 93–101 (2008)
3. Barequet, G., Eppstein, D., Goodrich, M.T., Vaxman, A.: Straight Skeletons of Three-Dimensional Polyhedra. In: Proc. 16th Annu. Europ. Symp. Algorithms, pp. 148–160 (September 2008)
4. Biedl, T., Held, M., Huber, S., Kaaser, D., Palfrader, P.: Weighted Straight Skeletons in the Plane. *Comput. Geom. Theory and Appl.* **48**(2), 120–133 (2015)
5. Eppstein, D., Erickson, J.: Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete Comput. Geom.* **22**(4), 569–592 (1999)
6. Fleiner, T., Irving, R.W., Manlove, D.F.: Efficient algorithms for generalized stable marriage and roommates problems. *Theoretical Computer Science* **381**(13), 162–176 (2007)
7. Haunert, J.-H., Sester, M.: Area Collapse and Road Centerlines Based on Straight Skeletons. *GeoInformatica* **12**, 169–191 (2008)
8. Huber, S.: *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*. Shaker Verlag (April 2012). ISBN: 978-3-8440-0938-5
9. Kelly, T., Wonka, P.: Interactive Architectural Modeling with Procedural Extrusions. *ACM Trans. Graph.* **30**(2), 14:1–14:15 (2011)
10. Laycock, R., Day, A.: Automatically Generating Large Urban Environments Based on the Footprint Data of Buildings. In: Proc. 8th Symp. Solid Modeling Applications, pp. 346–351 (June 2003)
11. Tan, J.J.: A Necessary and Sufficient Condition for the Existence of a Complete Stable Matching. *Journal of Algorithms* **12**(1), 154–178 (1991)
12. Tan, J.J., Hsueh, Y.-C.: A generalization of the stable matching problem. *Discrete Applied Mathematics* **59**(1), 87–102 (1995)

# Orienting Dynamic Graphs, with Applications to Maximal Matchings and Adjacency Queries

Meng He<sup>(✉)</sup>, Ganggui Tang, and Norbert Zeh

Faculty of Computer Science, Dalhousie University, Halifax, Canada  
{mhe, gtang, nzeh}@cs.dal.ca

**Abstract.** We consider the problem of edge orientation, whose goal is to orient the edges of an undirected dynamic graph with  $n$  vertices such that vertex out-degrees are bounded, typically by a function of the graph's arboricity. Our main result is to show that an  $O(\beta\alpha)$ -orientation can be maintained in  $O(\frac{\lg(n/(\beta\alpha))}{\beta})$  amortized edge insertion time and  $O(\beta\alpha)$  worst-case edge deletion time, for any  $\beta \geq 1$ , where  $\alpha$  is the maximum arboricity of the graph during update. This is achieved by performing a new analysis of the algorithm of Brodal and Fagerberg [2]. Not only can it be shown that these bounds are comparable to the analysis in Brodal and Fagerberg [2] and that in Kowalik [7] by setting appropriate values of  $\beta$ , it also presents tradeoffs that can not be proved in previous work. Its main application is an approach that maintains a maximal matching of a graph in  $O(\alpha + \sqrt{\alpha} \lg n)$  amortized update time, which is currently the best result for graphs with low arboricity regarding this fundamental problem in graph algorithms. When  $\alpha$  is a constant which is the case with planar graphs, for instance, our work shows that a maximal matching can be maintained in  $O(\sqrt{\lg n})$  amortized time, while previously the best approach required  $O(\lg n / \lg \lg n)$  amortized time [13]. We further design an alternative solution with worst-case time bounds for edge orientation, and applied it to achieve new results on maximal matchings and adjacency queries.

## 1 Introduction

The problem of orienting the edges of a dynamic undirected graph to guarantee a low upper bound on the maximum out-degree of its vertices has attracted much attention in recent years [2, 6, 7, 13]. In this problem, an orientation of a graph  $G = (V, E)$  is a directed graph  $\vec{G} = (V, \vec{E})$  defined by assigning each edge of  $G$  a direction, and  $\vec{G}$  is further called a  $\Delta$ -orientation if the out-degree of each vertex in  $\vec{G}$  is upper bounded by  $\Delta$ . The goal is to maintain a  $\Delta$ -orientation of  $G$  with efficient support of edge insertion and deletion, such that the value of  $\Delta$  is as small as possible. For dense graphs,  $\Delta$  has to be large, and thus this problem is more interesting when the graph is sparse.

As the arboricity of a graph is often used as a measurement of the sparsity of the graph, it is typically used as a parameter when bounding  $\Delta$ . The

---

This work is supported by NSERC and the Canada Research Chairs Program.



arboricity,  $\alpha$ , of a graph  $G$  can be formally defined by  $\alpha = \max_J \frac{|E(J)|}{|V(J)-1|}$ , where  $J = (V(J), E(J))$  is any subgraph of  $G$  induced by at least two vertices. Many classes of graphs in practice have constant arboricity, including planar graphs, graphs with bounded genus and graphs with bounded tree width. Nash-Williams [11, 12] proved that  $G$  has arboricity  $\alpha$  if and only if  $\alpha$  is the smallest number of subsets that  $E$  can be partitioned into, such that each subset of edges with their endpoints is a forest. Such a decomposition can be computed in polynomial time [3]. In this partition, if we orient each edge in a forest towards the root of the tree containing this edge, then each vertex has out-degree at most one in each forest, which immediately gives an  $\alpha$ -orientation of the given *static* graph.

The most fundamental application of edge orientation is perhaps the representation of dynamic graphs supporting adjacency queries. This is based on the following observation [5]: With a  $\Delta$ -orientation of  $G$ , if we store the at most  $\Delta$  out-neighbors of each vertex in a list, then an adjacency query can be answered in  $O(\Delta)$  time by scanning the list of each of the two vertices given in the query, to see if one is an out-neighbour of the other. Thus if we can maintain a  $\Delta$ -orientation of a sparse graph efficiently, then we immediately have a linear-space dynamic graph representation that answers adjacency queries in  $O(\Delta)$  time [2].

Recently, Neiman and Solomon [13] found that edge orientation also has applications in maintaining a maximal matching of a dynamic graph. A *matching*,  $M$ , of a graph  $G$  is a set of non-adjacent edges of  $G$ . If a matching  $M$  has the maximum number of edges, then it is called a *maximum cardinality matching*. A *maximal matching* is defined to be a matching,  $M$ , that satisfies the following condition: there does not exist an edge,  $g$ , of  $G$ , such that  $M \cup \{g\}$  is still a matching of  $G$ . It is well-known that any maximal matching is a 2-approximation for maximum cardinality matching. Graph matching is a fundamental problem in graph theory, and it has many applications in combinatorial optimization [10]. In the dynamic setting, the problem is to maintain a maximal matching or an approximate maximum cardinality matching under edge insertion and deletion. Recent progress on this [4, 6, 13] generated more interests in edge orientation.

Edge orientation has also been applied to other problems such as shortest path in dynamic planar graphs [6, 9] and graph colouring [8]. Motivated by all these important applications, we study the problem of orienting dynamic graphs.

## 1.1 Previous Work

Brodal and Fagerberg [2] first studied the problem of maintaining [2] an edge orientation of a dynamic graph with  $n$  vertices under an arboricity  $\alpha$  preserving sequence of edge insertions and deletions. Here an update operation is considered *arboricity  $\alpha$  preserving* if, when applied to an graph of arboricity at most  $\alpha$ , the arboricity of the graph after the update remains to be bounded by  $\alpha$ . They proposed an approach that can maintain an  $O(\alpha)$ -orientation using  $O(m+n)$  space, where  $m$  is the current number of edges, in  $O(1)$  amortized insertion time and  $O(\alpha + \lg n)$  amortized deletion time<sup>1</sup>. In their algorithm for update

<sup>1</sup> In this paper,  $\lg n$  denotes  $\log_2 n$ .

operations, some edges may change their orientation after each update, i.e., be *reoriented*. They proved that in terms of the amortized number of edge reorientations per update, their algorithm is  $O(1)$ -competitive compared against any algorithm. Kowalik [7] further showed that Brodal and Fagerberg’s approach can maintain an  $O(\alpha \lg n)$ -orientation with constant amortized insertion time and constant worst-case deletion time. More recently, Kopelowitz et al. [6] considered the problem of designing solutions to maintain edge orientation with worst-case time bounds. They showed how to maintain an  $O(\Delta)$ -orientation in  $O(m + n)$  space with  $O(\beta\alpha\Delta)$  worst-case insertion time and  $O(\Delta)$  worst-case deletion time, where  $\Delta \leq \inf_{\beta>1} \{\beta\alpha + \lceil \log_{\beta} n \rceil\}$ .

For maintaining matchings in arbitrary graphs, we refer to the recent work of Neiman and Solomon [13] which can maintain a maximal matching (which is also a  $3/2$ -approximate maximum cardinality matching) in  $O(\sqrt{m})$  worst-case update time, and the work of Gupta and Peng [4] which maintains a  $(1 + \epsilon)$ -approximation maximum cardinality matching with  $O(\sqrt{m}\epsilon^{-2})$  update time for any  $\epsilon > 0$ . To support more efficient updates for graphs with low arboricity, Neiman and Solomon [13] showed how to use edge orientation to maintain a maximal matching. Their approach can maintain a maximal matching in  $O(m + n)$  space, such that each update can be performed in  $O(\Delta + \log_{\Delta/\alpha} n)$  amortized time for any  $\Delta > 2\alpha$ . When  $\alpha = o(\lg n)$ , the update time becomes  $O(\frac{\lg n}{\lg((\lg n)/\alpha)} + \alpha)$ . Following the same idea, Kopelowitz et al. [6] made use of their solution for edge orientation to maintain a maximal matching, and the worst-case update time is asymptotically the same as their update time for maintaining edge orientation summarized in the previous paragraph.

As discussed previously, solutions to maintaining edge orientation can be directly used to represent dynamic graphs to support adjacency queries. Kowalik [7] showed that by maintaining the list of the out-neighbours of each vertex using the dynamic dictionary of Andersson and Thorup [1], a graph can be represented in  $O(m + n)$  space to support adjacency query and edge deletion in  $O(\lg \lg \lg n)$  worst-case time, and edge insertion in  $O(\lg \lg \lg n)$  amortized time, provided that  $\alpha = O(\text{polylog}(n))$ . Using the same strategy, Kopelowitz et al. [6] presented a linear-space representation of graphs with  $\alpha = \text{polylog}(n)$  arboricity that can support adjacency queries in  $O(\lg \lg \Delta)$  worst-case time, edge insertion in  $O(\beta\alpha\Delta \lg \lg \Delta)$  worst-case time, and edge deletion  $O(\Delta \lg \lg \Delta)$  worst-case time, where  $\Delta \leq \inf_{\beta>1} \{\beta\alpha + \lceil \log_{\beta} n \rceil\}$ .

## 1.2 Our Results

We first analyzed the algorithm of Brodal and Fagerberg [2], by constructing a new offline algorithm for their main reduction (summarized in Lemma 1). Our new analysis shows that an  $O(\beta\alpha)$ -orientations can be maintained in linear space with  $O(\frac{\lg(n/(\beta\alpha))}{\beta})$  amortized insertion time and  $O(\beta\alpha)$  worst-case deletion time, for any  $\beta \geq 1$ . Furthermore, no edge orientation is required when performing edge deletion. This presents a tradeoff between the maximum out-degree of vertices and insertion time in the analysis of the algorithm by Brodal and Fagerberg,

which was never proved before. If we set  $\beta = 1$ , then our analysis shows that this algorithm maintains an  $O(\alpha)$ -orientation while supporting insertion in  $O(\lg n)$  amortized time and deletion in  $O(\alpha)$  worst-case time. This is comparable to Brodal and Fagerberg's own analysis. By setting  $\beta = \lg n$ , the algorithm maintains an  $O(\alpha \lg n)$ -orientation with a constant number of edge reorientations per edge insertion in the amortized sense and zero reorientation for each deletion, which matches Kowalik [7]'s analysis.<sup>2</sup> When  $\beta = \sqrt{\lg n}$ , this algorithm maintains an  $O(\alpha\sqrt{\lg n})$ -orientation with  $O(\sqrt{\lg n})$  amortized insertion time and  $O(\alpha\sqrt{\lg n})$  worst-case deletion time. This tradeoff can not be shown using previous analysis.

We then apply our result on edge orientation to improve previous results on maintaining maximal matchings under arboricity  $\alpha$  preserving update sequences. More specifically, we can maintain a maximal matching using  $O(m + n)$  space in  $O(\alpha + \sqrt{\alpha \lg n})$  amortized update time, which is currently the best result on maintaining a maximal matching for low arboricity graphs. Our result matches the result of Neiman and Solomon [13] when  $\alpha = \Omega(\lg n)$ , while strictly improves their results when  $\alpha = o(\lg n)$ . To see the improvement when  $\alpha = o(\lg n)$ , suppose  $\alpha = \frac{\lg n}{f(n)}$ , where  $f(n)$  is an arbitrary function in  $\omega(1)$ . Then Neiman and Solomon's result supports updates in  $O(\frac{\lg n}{\lg f(n)})$  amortized time, while ours requires  $O(\frac{\lg n}{\sqrt{f(n)}})$ . The improvement is even obvious for graphs with constant arboricity such as planar graphs: a maximal matching can be maintained in  $O(\sqrt{\lg n})$  amortized time with our work, while previously it required  $O(\lg n / \lg \lg n)$  amortized time, and this improvement is surprising.

We further design solutions to these problems that guarantee worst-case time bounds. We show how to maintain a  $\Delta$ -orientation in  $O(\Delta)$  worst-case insertion and deletion time, where  $\Delta \leq 2\alpha \lg(n/\alpha) + 2\alpha$ . This is a new tradeoff when compared with the result of Kopelowitz et al. [6]: When  $\alpha = \omega(\lg n)$ , our insertion time is  $O(\alpha \lg n)$ , which is better than their  $O(\alpha^2)$  insertion time, though our maximum out-degree and deletion time are worse. It is noteworthy that our approach is simpler and does not require edge reorientation during insertion. The same bounds can be proved when applying our result to maintain a maximal matching, which again compares similarly to the result of Kopelowitz et al. We can also use this to represent a graph with  $O(\text{polylog}(n))$  arboricity to support adjacency queries in  $O(\lg \lg \Delta)$  worst-case time, edge insertion in  $O(\Delta)$  worst-case time, and edge deletion in  $O(\Delta \lg \lg \Delta)$  worst-case time. For graphs with constant arboricity such as planar graphs, our representation supports adjacency query, insertion and deletion in  $O(\lg \lg \lg n)$ ,  $O(\lg n)$  and  $O(\lg n \lg \lg \lg n)$  time, respectively, improving Kopelowitz et al.'s result which provides the same

<sup>2</sup> Kowalik [7]'s analysis in deletion time does not include the time required to find the location of the given edge within the list of out-neighbours of one of its endpoints and thus his model implicitly requires such a location to be given when performing deletion. In our work, unless otherwise specified, we follow the original model of Brodal and Fagerberg [2], which maintains out-neighbours in linked lists, and the time required to search each list for the edge to be deleted is part of deletion time. Thus when comparing with Kowalik's analysis, we consider the number of reorientations.

support for query and deletion, but requires  $O(\lg n \lg \lg \lg n)$  time for insertion. The fact that our insertion algorithm for edge orientation does not require reorientation makes such an improvement possible. For non-constant  $\alpha$ , our result is a new tradeoff: our insertion is faster than [6] but query and deletion may be slower. All our solutions use  $O(n + m)$  space.

## 2 Preliminaries

### 2.1 Reduction from Online Orientations to Offline Orientations

Brodal and Fagerberg [2] analyzed their algorithm by reducing the problem of maintaining an edge orientation under online updates to the problem of finding a sequence of orientations for an update sequence given offline. A variant of their reduction to be used in our solution can be summarized as:

**Lemma 1** ([2]). *Given an arbitrary arboricity  $\alpha$  preserving sequence of edge insertions and deletions over an initially empty graph, let  $G_0$  denote the initial empty graph,  $G_i$  denote the graph after the  $i$ th operation, and  $k$  denote the number of edge insertions.*

*If there exists a sequence  $\vec{G}_0, \vec{G}_1, \dots, \vec{G}_{p+q}$  of  $\delta$ -orientations that incurs at most  $kr$  edge reorientations in total for a certain  $r$ , then starting with the empty graph on  $n$  vertices under arbitrary arboricity  $\alpha$  preserving updates, a  $\Delta$ -orientation can be maintained using  $O(m + n)$  space, where  $m$  is the current number of edges, such that each edge insertion can be performed in  $O(\frac{r(\Delta+1)}{\Delta+1-2\delta})$  amortized time, and an edge deletion in  $O(\Delta)$  worst-case time, provided  $\Delta \geq 2\delta > 2\alpha$ . Furthermore, the amortized number of edge reorientations incurred during each insertion is  $O(\frac{r(\Delta+1)}{\Delta+1-2\delta})$ , and deletion requires no reorientation.*

### 2.2 Data Structures for Dynamic Sets with Center Elements

Kopelowitz et al. [6] defined the following data structure problem to help them maintain the invariants in their work, and we will also make use of this data structure in our solution with worst-case time bounds: Let  $X$  be a dynamic set, in which each element  $x_i \in X$  is associated with a nonnegative integer key  $k_i$ . The element  $x_0$  is designated as the center element of  $X$  which can not be inserted or deleted, but the value of its key can be updated. The goal is to support the following operations:

- **ReportMax**( $X$ ): return a pointer to an element in  $X$  with the maximum key;
- **Increment**( $X, x$ ): Given a pointer to  $x \in X \setminus \{x_0\}$ , increment  $x$ 's key;
- **Decrement**( $X, x$ ): Given a pointer to  $x \in X \setminus \{x_0\}$ , decrement  $x$ 's key;
- **Insert**( $X, x, k$ ): Insert a new element  $x$  with key  $k$  into  $X$ , provided  $k \leq k_0 + 1$ ;
- **Delete**( $X, x$ ): Given a pointer to  $x \in X \setminus \{x_0\}$ , remove  $x$  from  $X$ ;
- **IncrementCenter**( $X$ ): Increment  $k_0$ ;
- **DecrementCenter**( $X$ ): Decrement  $k_0$ .

The following lemma summarizes a solution to this problem:

**Lemma 2 ([6]).** *Let  $X$  be a dynamic set in which each element  $x_i$  is associated with a key  $k_i$  and a fixed element  $x_0$  is designated to be  $X$ 's center. Then  $X$  can be maintained in  $O(|X| + k_0)$  space to support **ReportMax**, **Increment**, **Decrement**, **Insert** and **Delete** in  $O(1)$  time, and **IncrementCenter** and **DecrementCenter** in  $O(k_0)$  time.*

### 3 Solutions with Amortized Time Bounds

In this section we first present a new offline algorithm to orient fully dynamic graphs. Then we make use of Lemma 1 to prove our result on maintaining edge orientation under online update operations.

In our offline strategy, let  $U$  be an arbitrary arboricity  $\alpha$  preserving update sequence on an initially empty graph  $G$  with  $n$  vertices. Denote by  $G_i$  the graph after the  $i$ th update as in Lemma 1 ( $G_0$  denotes the initial empty graph). We now show how to determine a sequence of  $\delta$ -orientations  $\vec{G}_0, \vec{G}_1, \dots, \vec{G}_U$  with a provable upper bound on the total number of edge reorientations, for a parameter  $\delta$  to be determined later. Note that it is trivial to orient the empty graph  $G_0$ .

We first divide  $U$  into *phases* each containing  $\beta\alpha n$  consecutive update operations, except the last phase which may contain fewer operations, where  $\beta \geq 1$ . For simplicity, we assume that  $\beta\alpha n$  is an integer. For the graph at the end of each phase that contains  $\beta\alpha n$  operations, we compute an  $\alpha$ -orientation using the approach described in the second paragraph of Section 1, which makes use of the algorithm in [3]. This determines the orientation of the graph at the end of each phase with the possible exception of the last phase, i.e.,  $\vec{G}_{\beta\alpha n}, \vec{G}_{2\beta\alpha n}, \vec{G}_{3\beta\alpha n}, \dots, \vec{G}_{\lfloor |U|/(\beta\alpha n) \rfloor (\beta\alpha n)}$ . To further orient  $G_i$  where  $i$  is not divisible by  $\beta\alpha n$ , we have the following definition:

**Definition 1.** *Consider a phase,  $P$ , of  $\beta\alpha n$  consecutive updates on a graph  $G$  with  $n$  vertices, in which an update operation that inserts or deletes an edge between vertices  $x$  and  $y$  is said to update  $x$  and  $y$ . A vertex of  $G$  is *hot* in  $P$  if it is updated by at least  $4\beta\alpha$  operations of  $P$ , and *cold* otherwise. The *hot region*,  $H(G)$ , of  $G$  in  $P$  is the subgraph of  $G$  induced by all the hot vertices of  $G$  in  $P$ , while the *cold region*,  $C(G)$ , of  $G$  in  $P$  is defined to be  $G \setminus H(G)$ .*

The  $\delta$ -orientation sequence is determined recursively. We use the following strategy for each phase,  $P$ , of  $U$ . Without loss of generality, we assume that  $|P| = \beta\alpha n$ . Let  $G_{i+j}$  denote the graph after the  $j$ th operation in  $P$ . Thus  $G_i$  denotes the graph immediately before any operation in  $P$  is performed, and by our previous discussion,  $\vec{G}_i$  is a  $\alpha$ -orientation of  $G$ . We determine the orientations of some of the edges in  $G_{i+j}$  for  $j \in [1.. \beta\alpha n - 1]$  in increasing order of  $j$ : For an edge that is present in both  $G_{i+j}$  and  $G_{i+j-1}$ , if its orientation in  $G_{i+j-1}$  has already been determined, then in  $G_{i+j}$ , we maintain the same orientation. There are no new edges to be oriented in  $G_{i+j}$  if the  $j$ th operation in  $P$  deletes an edge. If this

operation inserts an edge instead, then there are three cases. In the first case, this edge is between a hot vertex and a cold vertex, and we orient it from the cold vertex to the hot vertex. In the second case, the edge is between two cold vertices, and we orient it arbitrarily. In the remaining case, the edge is between two hot vertices, and we do not orient this edge in this level of recursion.

So far we have finished describing our top-level partition, which determines  $\vec{G}_i$  for  $i$  divisible by  $\beta\alpha n$ , and for all other  $G_i$ 's, it determines the orientations of the edges that are not inserted as an edge between two hot vertices during the phase containing this insertion. Then, for each phase,  $P$ , of  $U$ , let  $n'$  denote the number of vertices of  $G$  that are hot vertices in this phase. As each hot vertex is updated by at least  $4\beta\alpha$  operations in  $P$  and each operation may update up to two hot vertices, the number of operations in  $P$  that update hot vertices is at least  $2\beta\alpha n'$ . As this can not be larger than the total number of operations in  $P$ , we have  $2\beta\alpha n' \leq \beta\alpha n$ , which implies  $n' \leq n/2$ . If  $n' < 4\beta\alpha$ , we arbitrarily orient the edges inserted between these hot vertices by operations in phase  $P$  excluding the last operation (recall that after the last operation, the graph is oriented by computing an  $\alpha$ -orientation, so we exclude the last operation here). Otherwise, we set  $n$  to be  $n'$ , set  $U$  to be the sequence of operations in  $P$  that update hot vertices only, and apply the same recursive strategy to  $H(G)$ . Upon returning from the recursion on  $H(G)$ , the direction of each edge inserted between hot vertices have been decided as it is part of the graph  $H(G)$ . Thus we have oriented all the  $G_i$ 's. We now bound vertex out-degrees:

**Lemma 3.** *The offline algorithm in this section computes a sequence of  $(4\beta\alpha + \alpha)$ -orientations  $\vec{G}_0, \vec{G}_1, \dots, \vec{G}_{p+q}$ .*

*Proof.* We prove by induction that at each level of recursion, we construct  $(4\beta\alpha + \alpha)$ -orientations throughout each phase. In the base case where we stop the recursion, we consider a graph with at most  $4\beta\alpha$  vertices. In this case, even though we orient edges arbitrarily upon insertion, the maximum out-degree of any vertex is at most  $4\beta\alpha - 1$  as the total number of vertices is at most  $4\beta\alpha$ .

In the inductive case, for an arbitrary phase  $P$ , let  $G_{i+j}$  denote the graph after the  $j$ th operation in  $P$ . Assume inductively that the out-degree of any vertex in  $H(G)$  is at most  $4\beta\alpha + \alpha$  during the execution of the operations in  $P$ , and we now prove the same claim for  $G$ . We first consider an arbitrary cold vertex  $x$  in this phase. Before any operation in  $P$  is performed, in  $G_i$ , the out-degree of  $x$  is at most  $\alpha$  as  $\vec{G}_i$  is computed as an  $\alpha$ -orientation. By Definition 1, less than  $4\beta\alpha$  edges inserted in  $P$  have  $x$  as an endpoint. Thus the maximum out-degree of  $x$  in phase  $P$  is less than  $\alpha + 4\beta\alpha$ . We then argue about an arbitrary hot vertex  $y$ . As any edge between  $y$  and a cold vertex is oriented towards  $y$ , the out-degree of  $y$  is always equal to its out-degree in  $H(G)$ , which is bounded by  $4\beta\alpha + \alpha$  by inductive hypothesis.  $\square$

To bound the total number of edge reorientations, we have:

**Lemma 4.** *The total number of edge reorientations among  $\vec{G}_0, \vec{G}_1, \dots, \vec{G}_U$  is  $O\left(\frac{|U| \lg(n/(\beta\alpha))}{\beta}\right)$ .*

*Proof.* We number each level of recursion by its recursion depth starting from 0. Thus at level 0, we consider the original graph  $G$  with  $n$  vertices. At level 1, each of the subgraphs being considered corresponds to a phase at level 0 and contains the hot region of  $G$  in this phase which has at most  $n/2$  vertices, and so on. The number of vertices in each subgraph considered at level  $i$  is thus at most  $n/2^i$ , and the number of vertices of each graph considered at the last level is at most  $4\beta\alpha$ . Therefore, the number of levels is  $O(\lg(n/(\beta\alpha)))$  and the number of edges in each subgraph considered at level  $i$  is at most  $\alpha(n/2^i - 1)$ .

Note that at any given level, reorientation only happens at the end of each phase defined for a subgraph at that level, when we recompute an  $a$ -orientation and use it to orient the subgraph. We also observe that each operation in  $U$  may be considered at most once at each level of partition. As the number of levels is  $O(\lg(n/(\beta\alpha)))$ , it suffices to prove that, when amortizing the number of reorientations at the end of each phase at any level over the operations in that phase, the number of reorientations charged to each operation in this phase is at most  $1/\beta$ . To see this, let  $t$  denote the number of vertices in a subgraph considered at an arbitrary level. By our algorithm, the update sequence considered for this subgraph is divided into phases each containing  $\beta\alpha t$  operations, except the last phase which may contain fewer. Edge reorientations take place at the end of each phase that contains exactly  $\beta\alpha t$  operations. As the total number of edges in the subgraph is at most  $\alpha(t - 1)$ , the number of edge reorientations at the end of each such phase is thus at most  $\alpha(t - 1)$ . When amortizing these edge reorientations over the  $\beta\alpha t$  operations in the phase, each update is charged at most  $\alpha(t - 1)/(\beta\alpha t) < 1/\beta$  edge reorientations.  $\square$

Combining Lemmas 3 and 4, we have:

**Lemma 5.** *Given an arboricity  $\alpha$  preserving sequence of edge insertions and deletions on an initially empty graph and an arbitrary parameter  $\beta \geq 1$ , there is a sequence of  $(4\beta\alpha + \alpha)$ -orientations such that the amortized number of edge reorientation for each edge insertion or deletion is  $O(\frac{\lg(n/(\beta\alpha))}{\beta})$ .*

We now present our first main result:

**Theorem 1.** *Starting with the empty graph on  $n$  vertices under arboricity  $\alpha$  preserving updates, a  $\Delta$ -orientation can be maintained in  $O(n + m)$  space, where  $\Delta \geq 2\delta$ ,  $\delta = (4\beta + 1)\alpha$ ,  $\beta$  is an arbitrary parameter greater or equal to 1 and  $m$  is the current number of edges, such that an edge insertion can be performed in  $O(\frac{\lg(n/(\beta\alpha))}{\beta} \cdot \frac{\Delta + 1}{\Delta + 1 - 2\delta})$  amortized time, and an edge deletion in  $O(\Delta)$  worst-case time. Furthermore, edge deletion does not incur edge reorientation.*

*Proof.* As the graph is initially empty, the number,  $k$ , of insertions is greater than or equal to the number,  $k'$ , of deletions in  $U$ . Thus Lemma 5 shows that the total number of edge reorientations is  $O(\frac{(k+k') \lg(n/(\beta\alpha))}{\beta}) \leq O(\frac{2k \lg(n/(\beta\alpha))}{\beta}) = O(k \cdot (\frac{\lg(n/(\beta\alpha))}{\beta}))$ . The theorem thus follows from Lemma 1.  $\square$

The tradeoff summarized in Section 1.2 is obtained by setting  $\Delta = 3\delta$ . By applying this to maximal matchings, we have the following theorem:

**Theorem 2.** *Starting with the empty graph on  $n$  vertices under arboricity  $\alpha$  preserving updates, a maximal matching can be maintained in  $O(\alpha + \sqrt{\alpha \lg n})$  amortized time using  $O(n + m)$  space, where  $m$  is the current number of edges.*

*Proof.* Neiman and Solomon [13] made use of the algorithm of Brodal and Fagerberg [2] to maintain maximal matchings in dynamic settings. Their reduction shows that if a  $\Delta$ -orientation for a graph  $G$  on  $n$  vertices under arboricity  $\alpha$  preserving updates can be maintained in  $O(m + n)$  space with amortized update time  $T$ , where  $m$  denotes the current number of edges, then a maximal matching can also be maintained in  $O(m + n)$  space with  $O(\Delta + T)$  amortized update time.

We first observe that, according to Neiman and Solomon's reduction, a maximal matching can be maintained in  $O(\beta\alpha + \frac{\lg(n/(\beta\alpha))}{\beta})$  amortized update time using  $O(n + m)$  space, following from Theorem 1 by setting  $\Delta = 3\delta$ . When  $\alpha \geq \lg n$ , we set  $\beta = 1$  and the update time is  $O(\alpha)$ . Otherwise, we set  $\beta = \sqrt{\frac{\lg n}{\alpha}}$ , and the update time becomes  $O(\sqrt{\alpha \lg n})$ . The theorem thus follows.  $\square$

## 4 Solutions with Worst-Case Time Bounds

Let  $d_o(v)$  denote the out-degree of a vertex  $v$ . Our solution with worst-case time bounds maintains the following invariant over the entire graph  $G$  during updates:

**Invariant 1.** *For each vertex  $u$ , there exists an ordering of its out-neighbours,  $v_0, v_1, v_2, \dots, v_{d_o(u)-1}$ , such that  $d_o(v_i) \geq i$  for  $i = 0, 1, \dots, d_o(u) - 1$ .*

There are connections between this invariant and the invariants considered by Kopelowitz et al. [6], but they are different. The following two lemmas show why Invariant 1 can be used to bound the maximum vertex out-degree.

**Lemma 6.** *If the maximum out-degree,  $\Delta$ , of a vertex in a directed graph  $G$  of arboricity  $\alpha$  satisfying Invariant 1 is greater than  $4\alpha$ , then there are  $2^k\alpha$  vertices whose out-degrees are at least  $\Delta - 2k\alpha \geq 2\alpha$ , for  $k = 1, 2, \dots, \lfloor \Delta/(2\alpha) \rfloor - 1$ .*

*Proof.* The maximum value of  $k$  guarantees that  $\Delta - 2k\alpha \geq 2\alpha$ . To prove the rest of the lemma besides this inequality, let  $u$  be a vertex with out-degree  $\Delta$  in  $G$ . We prove our claim by induction on  $k$ . In the base case,  $k = 1$ . Let  $v_0, v_1, v_2, \dots, v_{\Delta-1}$  be  $u$ 's out-neighbours listed in the order specified in Invariant 1. Then  $d_o(v_{\Delta-1}) \geq \Delta - 1, d_o(v_{\Delta-2}) \geq \Delta - 2, \dots, d_o(v_{\Delta-2\alpha}) \geq \Delta - 2\alpha$  by Invariant 1, which means  $u$  has at least  $2\alpha$  out-neighbours with out-degrees greater than or equal to  $\Delta - 2\alpha$ .

Assume the claim holds for  $k - 1$ , and we prove it for  $k$ . By the inductive hypothesis, there is a set,  $V_1$ , of  $2^{k-1}\alpha$  vertices with out-degree at least  $\Delta - 2(k - 1)\alpha$ . By Invariant 1, each vertex in  $V_1$  has  $2\alpha$  out-neighbours whose out-degrees are at least  $\Delta - 2(k - 1)\alpha - 2\alpha = \Delta - 2k\alpha$ . We add such  $2\alpha$  out-neighbours of each vertex in  $V_1$  into another set  $V_2$ . Note that some vertices in  $V_1$  may share out-neighbors. Any vertex in  $V_1 \cup V_2$  has out-degree at least  $\Delta - 2k\alpha$ , and what remains is to give a lower bound on  $|V_1 \cup V_2|$ . Consider the subgraph  $G^*$  induced by  $V_1 \cup V_2$ . For each vertex in  $V_1$ , there are  $2\alpha$  distinct edges between it and the



vertices in  $V_2$ , and thus the number of edges in  $G^*$  is at least  $2\alpha|V_1| = 2^k\alpha^2$ . By the definition of arboricity, we have  $\alpha \geq \frac{|E(G^*)|}{|V(G^*)|-1} \geq \frac{2^k\alpha^2}{|V_1 \cup V_2|-1}$ . Therefore,  $|V_1 \cup V_2| \geq 2^k\alpha$ . As the out-degree of each vertex in  $V_1 \cup V_2$  is at least  $\Delta - 2k\alpha$  from our previous discussion, our induction goes through.  $\square$

**Lemma 7.** *If a directed graph  $G$  satisfies Invariant 1, then the out-degree of any vertex in  $G$  is at most  $2\alpha \lg(n/\alpha) + 2\alpha$ .*

*Proof.* Let  $\Delta$  denote the maximum out-degrees of the nodes in  $G$ . If  $\Delta \leq 4\alpha$ , the lemma holds because, in an undirected graph, we always have  $\alpha \leq n/2$  and thus  $2\alpha \lg(n/\alpha) + 2\alpha \geq 4\alpha$ . Otherwise, by Lemma 6, the number of vertices whose out-degrees are at least  $2\alpha$  is  $2^{\lfloor \Delta/(2\alpha) \rfloor - 1}\alpha$ . Therefore, the total number of edges of  $G$  is at least  $2^{\lfloor \Delta/(2\alpha) \rfloor - 1}\alpha \cdot 2\alpha = 2^{\lfloor \Delta/(2\alpha) \rfloor}\alpha^2$ . Since the arboricity of  $G$  is  $\alpha$ , we have  $(2^{\lfloor \Delta/(2\alpha) \rfloor}\alpha^2)/(n-1) \leq \alpha$ , and thus  $(2^{\Delta/(2\alpha)-1}\alpha^2)/(n-1) < \alpha$ . Therefore,  $\Delta < 2\alpha \lg \frac{n-1}{\alpha} + 2\alpha$ . This completes the proof.  $\square$

To maintain Invariant 1, we borrow ideas from [6] though our algorithms for edge insertion and deletion turn out to be simpler. As in [6], for each vertex  $u$ , we construct a data structure  $B_u$  to maintain information for its in-neighbours, which is further used to decide which edges should be reoriented. More precisely, for vertex  $u$ , we construct a dynamic set  $B_u$  whose center element is  $u$  itself, with  $d_o(u)$  as its key.  $X \setminus \{u\}$  then contains as elements all the in-neighbours of  $u$ , and the key for each such element is the out-degree of this in-neighbour. We then represent  $B_u$  using Lemma 2. Clearly all these auxiliary data structures use  $O(m+n)$  space in total, where  $m$  is the current number of edges in  $G$ .

We also construct the adjacency lists for  $G$  with edge orientations, by maintaining the out-going edges of each vertex in a doubly linked list. This also requires  $O(m+n)$  space. For each directed edge  $(u,v)$  in  $u$ 's list, we maintain a bidirectional pointer between this edge and  $u$ 's representation in  $B_v$ . With this, when our algorithm for edge deletion uses **ReportMax** to find an edge for reorientation, we can update adjacency lists in constant time. Such a construction is also required to make the approach in [6] work, though it was not mentioned explicitly. As it is trivial to maintain the adjacency lists with these pointers and the maintenance cost is subsumed by our final time bounds, we do not explicitly discuss how to update these lists in the rest of this section.

To insert an edge  $uv$ , assume without loss of generality that  $d_o(u) \leq d_o(v)$ . Then we orient the edge from  $u$  to  $v$ . It can be easily shown that with this strategy, Invariant 1 is maintained and no reorientation is required. We further update  $B_u$  using **IncrementCenter** and  $B_{u'}$  for each out-neighbour,  $u'$ , of  $u$ , using **Increment**. Algorithm 1 presents the pseudo code for edge insertion.

Algorithm 2 presents the pseudocode for edge deletion. It first removes the edge to be deleted in lines 2-3. After this, the out-degree of  $u$  is decreased by 1, and the only vertices for which Invariant 1 may not hold have to be in-neighbours of  $u$ . To find out whether the invariant is still maintained for all the in-neighbours of  $u$ , we locate the in-neighbour,  $v'$ , with the largest out-degree in line 4. If the test in the while statement at line 5 is false, then the invariant still holds for

---

**Algorithm 1.** Insert( $G, u, v$ )

---

- 1: {Assume without loss of generality that  $d_o(u) \leq d_o(v)$ }
  - 2: Orient edge  $(u, v)$  from  $u$  to  $v$
  - 3: **IncrementCenter**( $B_u$ )
  - 4: **Insert**( $B_v, u, d_o(u)$ )
  - 5: **for** each out-neighbour,  $u'$ , of  $u$  such that  $u' \neq v$  **do**
  - 6:   **Increment**( $B_{u'}, u$ )
- 

---

**Algorithm 2.** Deletion: Delete( $G, u, v$ )

---

- 1: {Assume without loss of generality that the edge  $uv$  is oriented towards  $v$ }
  - 2: Remove edge  $(u, v)$
  - 3: **Delete**( $B_v, u$ )
  - 4:  $v' \leftarrow$  **ReportMax**( $B_u$ )
  - 5: **while**  $d_o(u) < d_o(v') - 1$  **do**
  - 6:   Flip the orientation of edge  $(v', u)$  so that it is oriented from  $u$  to  $v'$
  - 7:   **Delete**( $B_u, v'$ )
  - 8:   **Insert**( $B_{v'}, u, d_o(u)$ )
  - 9:    $u \leftarrow v'$
  - 10:    $v' \leftarrow$  **ReportMax**( $B_u$ )
  - 11: **DecrementCenter**( $B_u$ )
  - 12: **for** each out-neighbour,  $v'$ , of  $u$  **do**
  - 13:   **Decrement**( $B_{v'}, u$ )
- 

any in-neighbour of  $u$ . Otherwise, it is possible (though not necessary) that the invariant is not maintained for  $v'$  and some other in-neighbours of  $u$ . To maintain the invariants for these vertices, we reverse the direction of the edge  $(v', u)$  in line 6, and update auxiliary data structures accordingly in lines 7-8. After this the out-degree of  $u$  becomes the same as its original out-degree before this edge deletion is performed, and thus the invariant can not be violated for any of its in-neighbours whose out-degree did not change. The only in-neighbour whose out-degree has been changed is  $v'$ , and it is easy to see that the invariant is also maintained for  $v'$  as a result of the above steps:  $v'$  lost one out-neighbour but its out-degree was also decreased by 1. Now the the only vertices for which Invariant 1 may not hold have to be in-neighbours of  $v'$ . For  $v'$ , we then repeat the same process that we applied to  $u$ . This process terminates in at most  $\Delta + 1$  iterations, because each time we iterate on a node whose out-degree is strictly greater than the node in the previous iteration and the maximum vertex out-degree is  $\Delta$ . From the description of this process, we can also claim that, after the while loop in lines 5-10 terminates, the invariant is maintained, and lines 11-13 make sure that all the auxiliary structures are up-to-date.

As our algorithms for edge insertion and deletion maintain Invariant 1, by Lemma 7, they can maintain a  $\Delta$ -orientation of  $G$  for  $\Delta = 2\alpha \lg(n/\alpha) + 2\alpha$ . To analyze the running time of these two operations, we first observe that each loop in the pseudocode of these two algorithms is iterated at most  $\Delta$  times. Then, applying Lemma 2, we claim that both operations require  $O(\Delta)$  time. Thus:

**Theorem 3.** *A  $\Delta$ -orientation of a graph on  $n$  vertices can be maintained in  $O(n + m)$  space, where  $\Delta \leq 2\alpha \lg(n/\alpha) + 2\alpha$ ,  $\alpha$  is the current arboricity and  $m$  is the current number of edges, such that an edge insertion or deletion can be performed in  $O(\Delta)$  worst-case time. Furthermore, an edge insertion does not incur edge reorientation, while a deletion incurs at most  $\Delta + 1$  reorientations.*

If we allow one reorientation in edge insertion, then we can bound  $\Delta$  by  $\min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{m})$  without affecting update times. We omit the details due to page limit. These results can then be easily applied to achieve new results on maximal matchings and adjacency queries in dynamic graphs:

**Theorem 4.** *A maximal matching of a graph on  $n$  vertices can be maintained in  $O(\min(\alpha \lg(n/\alpha), \sqrt{m}))$  worst-case update time using  $O(n + m)$  space, where  $\alpha$  is the current arboricity and  $m$  is the current number of edges.*

**Theorem 5.** *A graph with  $n$  vertices and  $m$  edges can be represented in  $O(m + n)$  space to support adjacency queries in  $O(\lg \lg \Delta)$  worst-case time, edge insertion in  $O(\Delta)$  worst-case time, and edge deletion in  $O(\Delta \lg \lg \Delta)$  worst-case time, where  $\Delta = O(\alpha \lg(n/\alpha))$  and  $\alpha$  is the current arboricity of the graph, provided  $\alpha = O(\text{polylog}(n))$ .*

**Acknowledgments.** We are grateful for Tsvi Kopelowitz for helpful discussions.

## References

1. Andersson, A., Thorup, M.: Tight(er) worst-case bounds on dynamic searching and priority queues. In: STOC, pp. 335–342 (2000)
2. Brodal, G.S., Fagerberg, R.: Dynamic representations of sparse graphs. In: Dehne, F., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 342–351. Springer, Heidelberg (1999)
3. Gabow, H.N., Westermann, H.H.: Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica* **7**(5&6), 465–497 (1992)
4. Gupta, M., Peng, R.: Fully dynamic  $(1 + \epsilon)$ -approximate matchings. In: FOCS, pp. 548–557 (2013)
5. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. *SIAM J. Discrete Math.* **5**(4), 596–603 (1992)
6. Kopelowitz, T., Krauthgamer, R., Porat, E., Solomon, S.: Orienting fully dynamic graphs with worst-case time bounds. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part II. LNCS, vol. 8573, pp. 532–543. Springer, Heidelberg (2014)
7. Kowalik, L.: Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.* **102**(5), 191–195 (2007)
8. Kowalik, L.: Fast 3-coloring triangle-free planar graphs. *Algorithmica* **58**(3), 770–789 (2010)
9. Kowalik, L., Kurowski, M.: Oracles for bounded-length shortest paths in planar graphs. *ACM Transactions on Algorithms* **2**(3), 335–363 (2006)

10. Lovász, L., Plummer, M.: Matching Theory. AMS Chelsea Publishing (1986)
11. Nash-Williams, C.S.J.A.: Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society* **36**(1), 445–450 (1961)
12. Nash-Williams, C.S.J.A.: Decomposition of finite graphs into forests. *Journal of the London Mathematical Society* **39**(1), 12 (1964)
13. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. In: *STOC*, pp. 745–754 (2013)

# Dynamic and Multi-Functional Labeling Schemes

Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Noy Rotbart<sup>(✉)</sup>

Department of Computer Science, University of Copenhagen, Universitetsparken 5,  
2100 Copenhagen, Denmark  
{soerend,knudsen,noyro}@di.ku.dk

**Abstract.** We investigate labeling schemes supporting adjacency, ancestry, sibling, and connectivity queries in forests. In the course of more than 20 years, the existence of  $\log n + O(\log \log n)$  labeling schemes supporting each of these functions was proven, with the most recent being ancestry [Fraigniaud and Korman, STOC '10]. Several multi-functional labeling schemes also enjoy lower or upper bounds of  $\log n + \Omega(\log \log n)$  or  $\log n + O(\log \log n)$  respectively. Notably an upper bound of  $\log n + 2 \log \log n$  for adjacency+siblings and a lower bound of  $\log n + \log \log n$  for each of the functions siblings, ancestry, and connectivity [Alstrup et al., SODA '03]. We improve the constants hidden in the  $O$ -notation, where our main technical contribution is a  $\log n + 2 \log \log n$  lower bound for connectivity+ancestry and connectivity+siblings.

In the context of dynamic labeling schemes it is known that ancestry requires  $\Omega(n)$  bits [Cohen, et al. PODS '02]. In contrast, we show upper and lower bounds on the label size for adjacency, siblings, and connectivity of  $2 \log n$  bits, and  $3 \log n$  to support all three functions. We also show that there exist no efficient dynamic adjacency labeling schemes for planar, bounded treewidth, bounded arboricity and bounded degree graphs.

## 1 Introduction

A labeling scheme is a method of distributing the information about the structure of a graph among its vertices by assigning short *labels*, such that a selected function on pairs of vertices can be computed using only their labels. The concept was introduced by Kannan, Naor and Rudich [1], and explored by a wealth of subsequent work [2–7].

Labeling schemes for trees have been studied extensively in the literature due to their practical applications in improving the performance of XML search engines. Indeed, XML documents can be viewed as labeled forests, and typical

---

Research partly supported by Mikkel Thorup's Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research carrier programme.

Research partly supported by the FNU project AlgoDisc - Discrete Mathematics, Algorithms, and Data Structures.

queries over the documents amount to testing classic properties such as adjacency, ancestry, siblings and connectivity between such labeled tree nodes [8]. In their seminal paper, Kannan et. al. [1] introduced labeling schemes using at most  $2 \log n$  bits<sup>1</sup> for each of the functions adjacency, siblings and ancestry. Improving these results have been motivated heavily by the fact that a small improvement of the label size may contribute significantly to the performance of XML search engines. Alstrup, Bille and Rauhe [3] established a lower bound of  $\log n + \log \log n$  for the functions siblings, connectivity and ancestry along with a matching upper bound for the first two. For adjacency, a  $\log n + O(\log^* n)$  labeling scheme was presented in [2]. A  $\log n + O(\log \log n)$  labeling scheme for ancestry was established only recently by Fraigniaud and Korman [4].

In most settings, it is the case that the structure of the graph to be labeled is not known in advance. In contrast to the *static* setting described above, a *dynamic* labeling scheme receives the tree as an online sequence of topological events. Cohen, Kaplan and Milo [9] considered *dynamic labeling schemes* where the encoder receives  $n$  leaf insertions and assigns unique labels that must remain unchanged throughout the labeling process. In this context, they showed a tight bound of  $\Theta(n)$  bits for any dynamic ancestry labeling scheme. We stress the importance of their lower bound by showing that it extends to routing, NCA, and distance as well. In light of this lower bound, Korman, Peleg and Rodeh [10] introduced dynamic labeling schemes where node re-label is permitted and performed by message passing. In this model they obtain a compact labeling scheme for ancestry, while keeping the number of messages small. Additional results in this setting include conversion methods for static labeling schemes [10,11], as well as specialized distance [11] and routing [12] labeling schemes. See [13] for experimental evaluation.

Considering the static setting, a natural question is to determine the label size required to support some, or all, of the functions. Simply concatenating the labels mentioned yield a  $O(\log n)$  label size, which is clearly undesired. Labeling schemes supporting multiple functions<sup>2</sup> were previously studied for adjacency and sibling queries. Alstrup et al. [3] proved a  $\log n + 5 \log \log n$  label size which was improved by Gavaille and Labourel [14] to  $\log n + 2 \log \log n$ . See Table 1 for a summary of labeling schemes for forests including the results of this paper.

## 1.1 Our Contribution

We contribute several upper and lower bounds for both dynamic and multi-functional labeling schemes. First, we observe that the naïve  $2 \log n$  adjacency, siblings and connectivity labeling schemes are suitable for the dynamic setting without the need of relabeling. We then present simple families of insertion sequences for which labels of size  $2 \log n$  are required, showing that in the dynamic setting the naïve labeling schemes are in fact optimal. The result is in contrast to the static case, where adjacency labels requires strictly fewer bits

<sup>1</sup> Throughout this paper we let  $\log n = \lceil \log_2 n \rceil$  unless stated otherwise.

<sup>2</sup> We refer to such labeling schemes as multi-functional labeling schemes.

**Table 1.** Upper and lower label sizes for labeling trees with  $n$  nodes (excluding additive constants). Routing is reported in the designer-port model [17] and NCA with no pre-existing labels [5]. Functions marked with \* denote non-unique labeling schemes, and bounds without a reference are folklore. Dynamic labeling schemes are all tight.

Function	Static Label Size	Static Lower Bound	Dynamic
Adjacency	$\log n + O(\log^* n)$ [2]	$\log n + 1$	$2 \log n$ (Th. 1)
Connectivity	$\log n + \log \log n$ [3]	$\log n + \log \log n$ [3]	$2 \log n$ (Th. 1)
Sibling	$\log n + \log \log n$ [15]	$\log n + \log \log n$ [3]	$2 \log n$ (Th. 1)
Ancestry	$\log n + 2 \log \log n$ [4]	$\log n + \log \log n$ [3]	$n$ [9]
AD/S	$\log n + 2 \log \log n$ [16]	$\log n + \log \log n$ [3]	$2 \log n$ (Th. 1)
C/S	$\log n + 2 \log \log n$ (Th. 5)	$\log n + 2 \log \log n$ (Th. 6)	$3 \log n$ (Th. 4)
C/AN	$\log n + 5 \log \log n$ (Th. 5)	$\log n + 2 \log \log n$ (Th. 7)	$n$ [9]
C/AD/S	$\log n + 3 \log \log n$ (Th. 5)	$\log n + 2 \log \log n$ (Th. 6)	$3 \log n$ (Th. 4)
Routing	$(1 + o(1)) \log n$ [6]	$\log n + \log \log n$ [3]	$n$ (Sec. 3)
NCA	$2.772 \log n$ [5]	$1.008 \log n$ [5]	$n$ (Sec. 3)
Distance	$1/2 \log^2 n$ [7]	$1/8 \log^2 n$ [7]	$n$ (Sec. 3)
Sibling*	$\log n$	$\log n$	$\log n$
Connectivity*	$\log n$	$\log n$	$\log n$
C/S*	$\log n + \log \log n$ (Th. 5)	$\log n + \log \log n$ (Th. 8)	$2 \log n$

than both sibling and connectivity. The labeling schemes also reveal an exponential gap between ancestry and the functions mentioned for the dynamic setting. In Sec. 3.3 we show a construction of simple lower bounds of  $\Omega(n)$  for adjacency labeling schemes on various important graph families.

In the context of multi-functional labeling schemes, we show first that  $3 \log n$  bits are necessary and sufficient for any dynamic labeling scheme supporting adjacency and connectivity. The paper’s main technical contribution lies in Th. 6, where we use a novel technique and prove a lower bound of  $\log n + 2 \log \log n$  for any unique labeling scheme supporting both connectivity and siblings/ancestry. This lower bound is preceded by a simple upper bound, proving that any labeling scheme of size  $S(n)$  growing faster than  $\log n$  can be altered to support connectivity as well by adding at most  $\log \log n$  bits. Note that in the case of connectivity and siblings the upper and lower bounds match. All omitted proofs appear in [18].

## 2 Preliminaries

A binary string  $x$  is a member of the set  $\{0, 1\}^*$ , and we denote its size by  $|x|$ , and the concatenation of two binary strings  $x, y$  by  $x \circ y$ . A *label assignment* for a tree  $T = (V, E)$  is a mapping of each  $v \in V$  to a bit string  $\mathcal{L}(v)$ , called the *label* of  $v$ . Given a tree  $T$  rooted in  $r$  with  $n$  nodes, and let  $u, v \in V$ . The function  $adjacency(v, u)$  returns **true** if and only if  $u$  and  $v$  are adjacent in  $T$ <sup>3</sup>,

<sup>3</sup> A node is adjacent to itself.

$ancestry(v, u)$  returns **true** if and only if  $u$  is on the path  $r \rightsquigarrow v$ ,  $siblings(v, u)$  returns **true** if and only if  $u$  and  $v$  have the same parent in  $T^4$ ,  $routing(v, u)$  returns an identifier of the edge connected to  $u$  on the path to  $v$ ,  $NCA(v, u)$  returns the label of the first node in common on the paths  $u \rightsquigarrow r$  and  $v \rightsquigarrow r$ , and  $distance(v, u)$  returns the length of the path from  $v$  to  $u$ . The functions mentioned previously are also defined for forests. Given a rooted forest  $F$  with  $n$  nodes, for any two nodes  $u, v$  in  $F$  the function  $connectivity(v, u)$  returns **true** if  $v$  and  $u$  are in the same tree in  $F$ .

Given a function  $f$  defined on sets of vertices, an  $f$ -labeling scheme for a family of graphs  $\mathcal{G}$  consists of an encoder and decoder. The *encoder* is an algorithm that receives a graph  $G \in \mathcal{G}$  as input and computes a label assignment  $e_G$ . If the encoder receives  $G$  as a sequence of topological events<sup>5</sup> the labeling scheme is *dynamic*. The *decoder* is an algorithm that receives any two labels  $\mathcal{L}(v), \mathcal{L}(u)$  and computes the query  $d(\mathcal{L}(v), \mathcal{L}(u))$ , such that  $d(\mathcal{L}(v), \mathcal{L}(u)) = f(v, u)$ . The *size* of the labeling scheme is the maximum label size. If for all graphs  $G \in \mathcal{G}$ , the label assignment  $e_G$  is an injective mapping, i.e. for all distinct  $u, v \in V(G)$ ,  $e_G(u) \neq e_G(v)$ , we say that the labeling scheme assigns *unique* labels. Unless stated otherwise, the labeling schemes presented are assumed to assign unique labels. Moreover, we allow the decoder to know the label size.

Let  $G$  be a graph in a family of graphs  $\mathcal{H}$  and suppose that an  $f$ -labeling scheme assigns a node  $v \in G$  the label  $\mathcal{L}(v)$ . If  $\mathcal{L}(v)$  does not appear in any of the label assignments for the other graphs in  $\mathcal{H}$ , we say that the label is *distinct* for the labeling scheme over  $\mathcal{H}$ . This notion will be useful in proving the lower bounds. All labeling schemes constructed in this paper require  $O(n)$  encoding time and  $O(1)$  decoding time under the assumption of a  $\Omega(\log n)$  word size RAM model. See [6] for additional details.

### 3 Dynamic Labeling Schemes

We first note that the lower bound for ancestry due to Cohen, et. al. also holds for NCA, since the labels computed by an NCA labeling scheme can decide ancestry: Given the labels  $\mathcal{L}(u), \mathcal{L}(v)$  of two nodes  $u, v$  in the tree  $T$ , return true if  $\mathcal{L}(u)$  is equal to the label returned by the original NCA decoder, and false otherwise. Similarly, suppose a labeling scheme for routing<sup>6</sup> assigns 0 as the port number on the path to the root. Given  $\mathcal{L}(u), \mathcal{L}(v)$  as before, return true if  $routing(\mathcal{L}(u), \mathcal{L}(v)) \neq 0$  and  $routing(\mathcal{L}(v), \mathcal{L}(u)) = 0$ . Peleg [19] proved that any  $f(n)$  distance labeling scheme can be converted to  $f(n) + \log(n)$  labeling scheme for NCA by attaching the depth of any node. Since the depth of a node inserted can not change in our dynamic setting, we conclude that the lower bound applies to distance up to additive  $O(\log n)$  factor.

<sup>4</sup> By this definition, a node is a sibling to itself.

<sup>5</sup> Cohen et al. defines such a sequence as a set of insertion of nodes into an initially empty tree, where the root is inserted first, and all other insertions are of the form “insert node  $u$  as a child of node  $v$ ”. We extend it to support “remove leaf  $u$ ”, where the root may never be deleted.

<sup>6</sup> Routing in the designer port model [17], in which this assumption is standard.



### 3.1 Upper Bounds

The following naïve adjacency labeling scheme was introduced by Kannan et al. [1]. Consider an arbitrary rooted tree  $T$  with  $n$  nodes. Enumerate the nodes in the tree with the numbers 0 through  $n - 1$ , and let, for each node  $v$ ,  $Id(v)$  be the number associated with  $v$ . Let  $parent(v)$  be the parent of a node  $v$  in the tree. The label of  $v$  is  $\mathcal{L}(v) = (Id(v) \circ Id(parent(v)))$ , and the root is labeled  $(0, 0)$ . Given the labels  $\mathcal{L}(v), \mathcal{L}(v')$  of two nodes  $v$  and  $v'$ , two nodes are adjacent if and only if either  $Id(parent(v)) = Id(v')$  or  $Id(parent(v')) = Id(v)$  but not both, so that the root is not adjacent to itself.

This is also a dynamic labeling scheme for adjacency with equal label size. Moreover, it is also both a static and dynamic labeling scheme for sibling, in which case, the decoder must check if  $Id(parent(v)) = Id(parent(v'))$ . A labeling scheme for connectivity can be constructed by storing the component number rather than the parent id. After  $n$  insertions, each label contains two parts, each in the range  $[0, n - 1]$ . Therefore, the label size required is  $2 \log n$ .

The labeling schemes suggested extend to larger families of graphs. In particular, the dynamic connectivity labeling scheme holds for the family of all graphs. The family of  $k$ -bounded degree graphs enjoys a similar dynamic adjacency labeling scheme of size  $(k + 1) \log n$ .

### 3.2 Lower Bounds

We show that  $2 \log n$  is a tight bound for any dynamic adjacency labeling scheme for trees. We denote by  $\mathcal{F}_n(k)$  an insertion sequence of  $n$  nodes, creating an *initial path* of length  $1 < k \leq n$ , followed by  $n - k$  *adjacent leaves* to node  $k - 1$  on the path. The family of all such insertions sequences is denoted  $\mathcal{F}_n$ . For illustration see Fig. 1.

**Lemma 1.** *Fix some dynamic labeling scheme that supports adjacency. For any  $1 < k < n$ ,  $\mathcal{F}_n(k)$  must contain at least  $n - k$  distinct labels for this labeling scheme over  $\mathcal{F}_n$ .*

*Proof.* The labels of  $\mathcal{F}_n(n)$  are set to  $P_1 \dots P_n$  respectively. Since the encoder is deterministic, and since every insertion sequence  $\mathcal{F}_n(k)$  first inserts nodes on the initial path, these nodes must be labeled  $P_1 \dots P_k$ . Let the labels of the adjacent leaves of such an insertion sequence be denoted by  $L_1^k \dots L_{n-k}^k$ .

Clearly,  $L_1^k \dots L_{n-k}^k$  must be different from  $P_1 \dots P_n$ , as the only other labels adjacent to  $P_{k-1}$  are  $P_{k-2}$  and  $P_k$ , which have already been used on the initial path. Consider now any node labeled  $L_i^j$  of  $\mathcal{F}_n(j)$  for  $j \neq k$ . Assume w.l.o.g that  $j > k$ . Such a node must be adjacent to  $P_{j-1}$  and *not* to  $P_{k-1}$ , as  $P_{k-1}$  is contained in the path to  $P_{j-1}$ . Therefore we must have  $L_i^j \notin \{L_1^k, \dots, L_{n-k}^k\}$ .  $\square$

Identical lower bounds are attained similarly for both sibling and connectivity.

**Theorem 1.** *Any dynamic labeling scheme supporting either adjacency, connectivity, or sibling requires at least  $2 \log n - 1$  bits.*

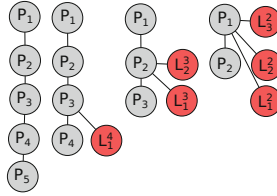


Fig. 1. Illustration of  $\mathcal{F}_5$

*Proof.* According to Lem. 1, at least  $n + \sum_{i=2}^{n-1} i = n^2/2 + O(n)$  distinct labels are required to label  $\mathcal{F}_n$  if adjacency or sibling requests are supported, and the same applies for  $\mathcal{F}_n^c$  if connectivity is supported.  $\square$

A natural question is whether a randomized labeling scheme could provide labels of size less than  $2 \log n - O(1)$ . The next theorem, based on Thm. 3.4 in [9] answer this question negatively.

**Theorem 2.** *For any randomized dynamic labelling scheme supporting either adjacency, connectivity, or sibling queries there exists an insertion sequence such that the expected value of the maximal label size is at least  $2 \log n - O(1)$  bits.*

### 3.3 Other Graph Families

In this section, we expand our lower bound ideas to adjacency labeling schemes for the following families with at most  $n$  nodes: bounded arboricity- $k$  graphs<sup>7</sup>  $\mathcal{A}_k$ , bounded degree- $k$  graphs  $\Delta_k$ , planar graphs  $\mathcal{P}$  and bounded treewidth- $k$  graphs  $\mathcal{T}_k$ . In the context of (static) adjacency labeling schemes, these families are well studied [1, 2, 20, 21]. In particular,  $\mathcal{T}_k$ ,  $\mathcal{P}$ ,  $\Delta_k$  and  $\mathcal{A}_k$  enjoy adjacency labeling schemes of size  $\log n + O(k \log \log(n/k))$  [20],  $2 \log n + O(\log \log n)$  [20],  $\lfloor \frac{\Delta(n)}{2} \rfloor + 1$  [21], and  $k \log n$  [21] respectively.

We consider a sequence of node insertions along with all edges adjacent to them, such that an edge  $(u, v)$  may be introduced along with node  $v$  if node  $u$  appeared prior in the sequence, and prove the following.

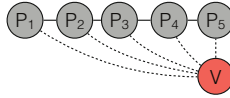
**Theorem 3.** *Any dynamic adjacency labeling scheme for each  $\mathcal{A}_2$ ,  $\mathcal{P}$  and  $\mathcal{T}_3$  requires  $\Omega(n)$  bits. Similarly, any dynamic adjacency labeling scheme for  $\Delta_k$  requires  $k \log n$  bits.*

*Proof.* Let  $S$  be the collection of all nonempty subsets of the integers  $1 \dots n-1$ . For every  $s \in S$ , we denote by  $\mathcal{F}_n(s)$  an insertion sequence of  $n$  nodes, creating a path of length  $n-1$ , followed by a single node  $v$  connected to the nodes on the path whose number is a member of  $s$ . Such a graph has arboricity 2 since it can be decomposed into an initial path and a star rooted in  $v$ . For each of the  $|S|$  insertion sequences,  $v$ 's label must be distinct. We conclude that the number of

<sup>7</sup> The *arboricity* of a graph  $G$  is the minimum number of edge-disjoint acyclic subgraphs whose union is  $G$ .

bits required for any adjacency labeling scheme is at least  $\log(|S|) = n - 1$ . See Fig. 2 for illustration.

The construction of  $\mathcal{F}_n(s)$  implies an identical lower bound for the family of planar graphs, as well as interval graphs. By considering all sets  $s$  of at most  $k$  elements instead, we get a bound of  $k \log n$  label size for any adjacency labeling scheme for  $\Delta_k$ , where  $k$  is constant.  $\square$



**Fig. 2.** Illustration of  $\mathcal{F}_5(s)$ . The dotted lines may or may not appear in the insertion sequence depending on the element of  $S$  chosen.

## 4 Multi-Functional Labeling Schemes

In this section we investigate labeling schemes incorporating two or more of the functions mentioned for both dynamic and static labeling schemes.

### 4.1 Dynamic Multi-functional Labeling Schemes

A  $3 \log n$  dynamic labeling scheme for any combination of connectivity, adjacency and sibling queries can be obtained by setting the label of a node  $v$  to be  $(Id(v) \circ Id(parent(v)) \circ component(v))$ , as described in Sec. 3.1.

We now show that this upper bound is in fact tight. More precisely, we show that  $3 \log n$  bits are required to answer the combination of connectivity and adjacency. Let  $I_n(j, k)$  be an insertion sequence designed as follows: First  $j$  nodes are inserted creating an *initial forest* of single node trees. Then  $k$  nodes are added as a path with root in the  $j$ th tree. At last,  $n - j - k$  adjacent *path leaves* are added to the second-to-last node on the path. For a given  $n$  we define  $I_n$  as the family of all such insertion sequences.

**Lemma 2.** *Fix some dynamic labeling scheme that supports adjacency and connectivity requests. For any  $1 < j + k < n$ ,  $I_n(k)$  must contain at least  $n - j - k$  distinct labels for this labeling scheme over  $I_n$ .*

According to this Lemma, at least  $\sum_{j=1}^{n-1} \sum_{k=1}^{n-j-1} n - j - k = \frac{1}{6}n^3 - O(n^2)$  distinct labels are required to label the family  $I_n$ . We can thus conclude.

**Theorem 4.** *Any dynamic labeling scheme supporting both adjacency and connectivity queries requires at least  $3 \log n - O(1)$  bits.*

The same family of insertion sequences can be used to show a  $3 \log n - O(1)$  lower bound for any dynamic labeling scheme supporting both sibling and connectivity queries. Furthermore, similarly to Thm. 2, the bound holds even without the assumption that the encoder is deterministic.

## 4.2 Upper Bounds for Static Multi-functional Labeling Schemes

As seen in Thm. 4, the requirement to support both connectivity and adjacency forces an increased label size for any dynamic labeling scheme. In the remainder of the paper we prove lower and upper bounds for static labeling schemes that support those operations, both for the case where the labels are necessarily unique, and for the case that they are not. From hereon, all labeling schemes are on the family of rooted forests with at most  $n$  nodes. We show that most labeling schemes can be altered to support connectivity as well.

**Theorem 5.** *Consider any function  $f$  of two nodes in a single tree on  $n$  nodes. If there exists an  $f$ -labeling scheme of size  $S(n)$ , where  $S(n)$  is non-decreasing and  $S(a) - S(b) \geq \log a - \log b - O(1)$  for any  $a \geq b$ . Then there exists an  $f$ -labeling scheme, which also supports connectivity queries of size at most  $S(n) + \log \log n + O(1)$ .*

*Proof.* We will consider the label  $\mathcal{L}(v) = (C \circ L \circ sep)$  defined as follows. First, sort the trees of the forest according to their sizes. For the  $i$ th biggest tree we set  $C = i$  using  $\log i$  bits. Since the tree has at most  $n/i$  nodes, we can pick the label  $L$  internally in the tree using only  $S(n/i)$  bits. Finally, we need a separator,  $sep$ , to separate  $C$  from  $L$ . We can represent this using  $\log \log n$  bits, since  $i$  uses at most  $\log n$  bits.

The total label size is  $\log i + S(n/i) + \log \log n + O(1)$  bits, which is less than  $S(n) + \log \log n + O(1)$  if  $S(n) - S(n/i) \geq \log i - c$  for some constant  $c$ . Since  $f$  is a function of two nodes from the same tree, this altered labeling scheme can answer both queries for  $f$  as well as connectivity. It is now required that any label assigned has size exactly  $S(n) + \log \log n$  bits, so that the decoder may correctly identify  $sep$  in the bit string. For that purpose we pad labels with less bits with sufficiently many 0's. The decoder can identify  $C$  in  $O(1)$  time.  $\square$

As a corollary, we get labeling schemes of the sizes reported in Table 1.

## 4.3 Lower Bounds for Static Multi-functional Labeling Schemes

We now show that the upper bounds implied by Thm. 5 for labeling schemes supporting siblings and connectivity are indeed tight for both the unique and non-unique cases. To that end we consider the following forests: For any integers  $a, b, n$  such that  $ab \mid n$  denote by  $F_n(a, b)$  a forest consisting of  $a$  components (trees), each with  $b$  sibling groups, where each sibling group consist of  $\frac{n}{a \cdot b}$  nodes. Note that  $n \leq |F_n(a, b)| < 2n$  since we add one auxiliary root per component.

Our proofs work as follows: Firstly, for any two forests  $F_n(a, b)$  and  $F_n(c, d)$  as defined above, we establish an upper bound on the number of labels that can be assigned to both  $F_n(a, b)$  and  $F_n(c, d)$ . Secondly, for a carefully chosen family of forests  $F_n(a_1, b_1), \dots, F_n(a_k, b_k)$ , we show that when labeling  $F_n(a_i, b_i)$  at least a constant fraction of the labels has to be distinct from the labels of  $F_n(a_1, b_1), \dots, F_n(a_{i-1}, b_{i-1})$ . Finally, by summing over each  $F_n(a_i, b_i)$  we show

that a sufficiently large number of bits are required by any labeling scheme supporting the desired queries.

Our technique simplifies the boxes and groups argument of Alstrup et al. [3], and generalizes to the case of two nested equivalence classes<sup>8</sup>, namely connectivity and siblings.

**Lemma 3.** *Let  $F_n(a, b)$  and  $F_n(c, d)$  be two forests such that  $ab \geq cd$ . Fix some unique labeling scheme supporting both connectivity and siblings, and denote the set of labels assigned to  $F_n(a, b)$  and  $F_n(c, d)$  as  $e_1$  and  $e_2$  respectively. Then*

$$|e_1 \cap e_2| \leq \min(a, c) \cdot \min(b, d) \cdot \frac{n}{a \cdot b} .$$

*Proof.* Consider label sets  $s_1$  and  $s_2$  of two sibling groups from  $F_n(a, b)$  and  $F_n(c, d)$  respectively for which  $|s_1 \cap s_2| \geq 1$ . Clearly, we must have  $|s_1 \cap s_2| \leq \min(|s_1|, |s_2|) = \frac{n}{a \cdot b}$ . Furthermore, no other sibling group of  $F_n(a, b)$  or  $F_n(c, d)$  can be assigned labels from  $s_1 \cup s_2$ , as the sibling relationship must be maintained. We can thus create a one-to-one matching between the sibling groups of  $F_n(a, b)$  and  $F_n(c, d)$ , that have labels in common (note that not all sibling groups will necessarily be mapped). Bounding the number of common labels thus becomes a problem of bounding the size of this matching. In order to maintain the connectivity relation, sibling groups from one component cannot be matched to several components. Therefore at most  $\min(b, d)$  sibling groups can be shared per component, and at most  $\min(a, c)$  components can be shared. Combining this gives the final bound of  $\min(a, c) \cdot \min(b, d) \cdot \frac{n}{a \cdot b}$ .  $\square$

**Lemma 4.** *Let  $F_n(a_1, b_1), \dots, F_n(a_i, b_i)$  be a family of forests with  $a_1 \cdot b_1 \leq \dots \leq a_i \cdot b_i$ . Assume there exists a unique labeling scheme supporting both connectivity and siblings, and let  $e_j$  be the set of labels assigned by this scheme to the forest  $F_n(a_j, b_j)$ . Assume that the sets  $e_1, \dots, e_{i-1}$  have been assigned. The number of distinct labels introduced by the encoder when assigning  $e_i$  is at least*

$$n - \sum_{j=1}^{i-1} \min(a_j, a_i) \cdot \min(b_j, b_i) \cdot \frac{n}{a_i \cdot b_i} .$$

We demonstrate the use of Lem. 4 by showing the following known result [3].

**Warm-up.** *Any static labeling scheme for connectivity queries requires at least  $\log n + \log \log n - O(1)$  bits.*

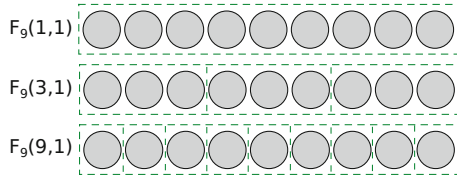
*Proof.* Consider the family of  $\log_3 n$  forests  $F_n(3^0, 1), F_n(3^1, 1), \dots, F_n(3^{\log_3 n}, 1)$ . This family is demonstrated in Fig. 3 for  $n = 9$ . Two nodes are siblings if and only if they are connected in this family. Therefore we can use Lem. 4 even though we want to show a lower bound for only connectivity. Note, that in Fig. 3 the second forest can at most reuse 3 labels from the first, and the third can at most reuse 4 from the two previous.

<sup>8</sup> See [15] for definitions and further discussion.

Let  $e_j$  denote the label set assigned by an encoder for  $F_n(3^j, 1)$ . We assume that the labels are assigned in the order  $e_0, \dots, e_{\log_3 n}$ . By Lem. 4 the number of distinct labels introduced when assigning  $e_j$  is at least

$$n - n \sum_{i=0}^{j-1} 3^{i-j} > n/2 .$$

It follows that labeling the  $\log_3 n$  forests in the family requires at least  $\Omega(n \log n)$  distinct labels. □



**Fig. 3.** The family of forests  $F_9(1, 1), F_9(3, 1), F_9(9, 1)$ . Nodes inside the same box are connected and siblings. Note that component roots have been omitted.

We are now ready to prove the main theorem of this section.

**Theorem 6.** *Any unique static labeling scheme supporting both connectivity and sibling queries requires labels of size at least  $\log n + 2 \log \log n - O(1)$ .*

*Proof.* Fix some integer  $x$ , and assume that  $n$  is a power of  $x$ . We consider the family of forests  $F_n(1, 1), F_n(x, 1), F_n(1, x), F_n(x^2, 1), F_n(x, x), F_n(1, x^2), \dots, F_n(1, x^{\log_x n})$ .

Let  $e_a^b$  denote the label set assigned to  $F_n(x^a, x^b)$  by an encoder. We assign the labels in the order  $e_0^0, e_1^0, e_0^1, e_2^0, e_1^1, \dots, e_{\log_x n}^{\log_x n}$ . Thus, when assigning  $e_a^b$  we have already assigned all label sets  $e_c^d$  with  $c + d < a + b$  or  $c + d = a + b$  and  $d < b$ . By Lem. 4, the number of distinct labels introduced when assigning  $e_a^b$  is at least

$$n - \sum_{\substack{c+d < a+b \\ c, d \geq 0}} \frac{n}{x^{a+b}} \cdot x^{\min(a,c) + \min(b,d)} + \sum_{d=0}^{b-1} \frac{n}{x^{a+b}} \cdot x^{a+d}$$

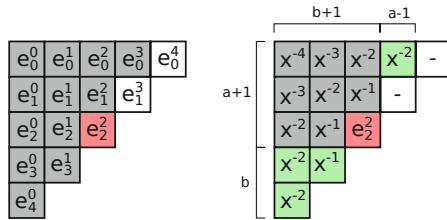
This counting argument is better demonstrated in Fig. 4. In the figure, we are concerned with assigning the labels in  $e_2^2$ . The grey boxes represent the label sets already assigned, and the right-side figure shows the fractions of  $n$  that each set  $e_c^d$  at most has in common with  $e_2^2$ . Observe that we can split the above sum into three cases as demonstrated in the figure: If  $c \leq a$  and  $d \leq b$  the bound supplied by Lem. 3 is  $x^{c+d-a-b}$ . Otherwise, either  $c > a$  or  $d > b$ , but not both. If  $c > a$ , recall that  $d < b$  so the bound is  $x^{d-b}$ . For  $d > b$  the bound is  $x^{c-a}$

by the same argument. Applying these rules, we see that the number of distinct labels introduced is at least

$$\begin{aligned}
 & n - n \cdot \left( \sum_{c=0}^a \sum_{d=0}^b x^{c+d-a-b} + \sum_{d=0}^{b-1} (b-d) \cdot x^{d-b} + \sum_{c=0}^{a-2} (a-c) \cdot x^{c-a} \right) + n \\
 & \geq n - n \cdot \left( \frac{x^2 + x + 2}{(x-1)^2} \right) + n = n - n \cdot \frac{3x + 1}{(x-1)^2}.
 \end{aligned}$$

Note that we add  $n$ , as we have also subtracted  $n$  labels for the case  $(c, d) = (a, b)$ .

By setting  $x = 6$  we get that the encoder must introduce  $6n/25$  distinct labels for each  $e_a^b$ . Since we have  $\Theta(\log^2 n)$  forests, a total of  $\Omega(n \log^2 n)$  labels are required for labeling the family of forests. Each forest consists of no more than  $2n$  nodes, which concludes the proof.  $\square$



**Fig. 4.** Demonstration of the label counting for  $e_2^2$

The same proof technique is used to prove the following theorems.

**Theorem 7.** Any unique static labeling scheme supporting both connectivity and ancestry queries requires labels of size at least  $\log n + 2 \log \log n - O(1)$ .

**Theorem 8.** Any static labeling scheme supporting both connectivity and sibling queries requires at least  $\log n + \log \log n - O(1)$  bits if the labels need not be unique.

*Proof.* Assume w.l.o.g. that  $n$  is a power of 3. Consider the family of  $\log_3 n$  forests  $F_n(1, n), F_n(3, n/3), F_n(3^2, n/3^2), \dots, F_n(3^{\log_3 n}, 1)$ . Since each sibling group of the forest  $F_n(3^i, n/3^i)$  has exactly one node, we note that no two nodes are siblings. Thus each label of the forest has to be unique, since we have assumed that a node is sibling to itself. We can thus use Lem. 3 as if we were in the unique case for this family of forests.

Let  $e_j$  denote the label set assigned by an encoder for  $F_n(3^j, n/3^j)$ . We assume that the labels are assigned in the order  $e_0, \dots, e_{\log_3 n}$ . By Lem. 4 the number of distinct labels introduced when assigning  $e_j$  is at least

$$n - n \sum_{i=0}^{j-1} 3^{i-j} > n/2.$$

It follows that when labeling each of the  $\log_3 n$  forests in the family, any encoder must introduce at least  $n/2$  distinct labels, i.e.  $\Omega(n \log n)$  distinct labels in total. The family consist of forests with no more than  $2n$  nodes, which concludes the proof.  $\square$

## 5 Concluding Remarks

We have considered multi-functional labels for the functions adjacency, siblings and connectivity. We also provided a lower bound for ancestry and connectivity. A major open question is whether it is possible to have a label of size  $\log n + O(\log \log n)$  supporting all of the functions. It seems unlikely that the best known labeling scheme for ancestry [4] can be combined with the ideas of this paper.

In the context of dynamic labeling schemes, if arbitrary insertion is permitted, neither adjacency nor sibling labels are possible. All dynamic labeling schemes also operate when leaf removal is allowed, simply by erasing the removed label.

## References

1. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. *SIAM Journal On Discrete Mathematics*, 334–343 (1992)
2. Alstrup, S., Rauhe, T.: Small induced-universal graphs and compact implicit graph representations. In: *FOCS 2002*, pp. 53–62 (2002)
3. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. *SIAM J. Discret. Math.* **19**(2), 448–462 (2005)
4. Fraigniaud, P., Korman, A.: An optimal ancestry scheme and small universal posets. In: *STOC 2010*, pp. 611–620 (2010)
5. Alstrup, S., Halvorsen, E.B., Larsen, K.G.: Near-optimal labeling schemes for nearest common ancestors. In: *SODA*, pp. 972–982 (2014)
6. Thorup, M., Zwick, U.: Compact routing schemes. In: *SPAA 2001*, pp. 1–10 (2001)
7. Peleg, D.: Proximity-preserving labeling schemes. *Journal of Graph Theory* **33**(3), 167–176 (2000)
8. Wu, X., Lee, M.L., Hsu, W.: A prime number labeling scheme for dynamic ordered xml trees. In: *Proceedings of the 20th International Conference on Data Engineering*, pp. 66–78. *IEEE* (2004)
9. Cohen, E., Kaplan, H., Milo, T.: Labeling dynamic xml trees. *SIAM Journal on Computing* **39**(5), 2048–2074 (2010)
10. Korman, A., Peleg, D., Rodeh, Y.: Labeling schemes for dynamic tree networks. *Theory of Computing Systems* **37**(1), 49–75 (2004)
11. Korman, A.: General compact labeling schemes for dynamic trees. *Distributed Computing* **20**(3), 179–193 (2007)
12. Korman, A.: Improved compact routing schemes for dynamic trees. In: *PODC 2008*, pp. 185–194. *ACM* (2008)
13. Rotbart, N., Vaz Salles, M., Zotos, I.: An evaluation of dynamic labeling schemes for tree networks. In: Gudmundsson, J., Katajainen, J. (eds.) *SEA 2014*. LNCS, vol. 8504, pp. 199–210. Springer, Heidelberg (2014)
14. Gavouille, C., Labourel, A.: Distributed relationship schemes for trees. In: Tokuyama, T. (ed.) *ISAAC 2007*. LNCS, vol. 4835, pp. 728–738. Springer, Heidelberg (2007)



15. Lewenstein, M., Munro, J.I., Raman, V.: Succinct data structures for representing equivalence classes. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) *Algorithms and Computation*. LNCS, vol. 8283, pp. 502–512. Springer, Heidelberg (2013)
16. Dahlgaard, S., Knudsen, M.B.T., Rotbart, N.: Improved ancestry labeling scheme for trees, arXiv preprint [arXiv:1407.5011](https://arxiv.org/abs/1407.5011)
17. Fraigniaud, P., Gavoille, C.: Routing in trees. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) *ICALP 2001*. LNCS, vol. 2076, pp. 757–772. Springer, Heidelberg (2001)
18. Dahlgaard, S., Knudsen, M.B.T., Rotbart, N.: Dynamic and multi-functional labeling schemes, arXiv preprint [arXiv:1404.4982](https://arxiv.org/abs/1404.4982)
19. Peleg, D.: Informative labeling schemes for graphs. *Theor. Comput. Sci.* **340**(3), 577–593 (2005)
20. Gavoille, C., Labourel, A.: Shorter implicit representation for planar graphs and bounded treewidth graphs. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) *ESA 2007*. LNCS, vol. 4698, pp. 582–593. Springer, Heidelberg (2007)
21. Adjiashvili, D., Rotbart, N.: Labeling schemes for bounded degree graphs. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) *ICALP 2014, Part II*. LNCS, vol. 8573, pp. 375–386. Springer, Heidelberg (2014)

# **Data Structures and Algorithms I**

# Hashing and Indexing: Succinct Data Structures and Smoothed Analysis

Alberto Policriti<sup>1,2</sup> and Nicola Prezza<sup>1</sup> (✉)

<sup>1</sup> Department of Mathematics and Informatics,  
University of Udine, Udine, Italy  
[prezza.nicola@spes.uniud.it](mailto:prezza.nicola@spes.uniud.it)

<sup>2</sup> Istituto di Genomica Applicata, Udine, Italy

**Abstract.** We consider the problem of indexing a text  $T$  (of length  $n$ ) with a light data structure that supports efficient search of patterns  $P$  (of length  $m$ ) allowing errors under the Hamming distance. We propose a hash-based strategy that employs two classes of hash functions—dubbed Hamming-aware and de Bruijn—to drastically reduce search space and memory footprint of the index, respectively.

We use our succinct hash data structure to solve the  $k$ -mismatch search problem in  $2n \log \sigma + o(n \log \sigma)$  bits of space with a randomized algorithm having smoothed complexity  $\mathcal{O}((2\sigma)^k (\log n)^k (\log m + \xi) + (occ + 1) \cdot m)$ , where  $\sigma$  is the alphabet size,  $occ$  is the number of occurrences, and  $\xi$  is a term depending on  $m$ ,  $n$ , and on the amplitude  $\epsilon$  of the noise perturbing text and pattern. Significantly, we obtain that for any  $\epsilon > 0$ , for  $m$  large enough,  $\xi \in \mathcal{O}(\log m)$ : our results improve upon previous linear-space solutions of the  $k$ -mismatch problem.

## 1 Introduction

Indexing is a very efficient choice when one is interested in rapidly searching and/or retrieving from a text all the occurrences of a large number of patterns. In particular, indexing large texts for inexact pattern matching is a problem that is lately receiving much attention, due to the continuously increasing rate at which data is produced in areas such as bioinformatics and web information retrieval, where, moreover, is critical to allow (a limited amount of) mismatches while searching. Techniques based on the Burrows-Wheeler transform are the gold standard when dealing with large text indexing; however, BWT-based indexes offer a natural support for exact string matching only. As a consequence, to deal with inexact search, simple space efficient strategies such as *backtracking*, *q-grams sampling*, and *hybrid* techniques are usually employed. Letting  $m$ ,  $k$ , and  $\sigma$  be the query length, the maximum number of allowed errors and the alphabet size, respectively, backtracking techniques have the disadvantage that query times rapidly blow-up with a factor of  $\sigma^k m^k$  and are thus impractical for large patterns and number of errors (a backtracking strategy on the FM index is implemented in the tool Bowtie [1]). q-gram based strategies do not suffer of this exponential blow-up but their usage is limited to a small number of errors,

due to the fact that q-grams are searched without errors (SOAP2 [2] implements a q-gram strategy on the FM index). Hybrid strategies combine the two approaches and are often able to obtain better time bounds without restrictions. An example of this kind (yet requiring  $\Theta(n \log n)$  bits of space, where  $n$  is the text length) is the hash-based algorithm rNA presented in [3,4] which employs the notion of *Hamming-aware* hash function, to be discussed below. Other solutions in the literature reach a time complexity often *linear* in the query length, at the price of significant space consumption. Letting  $n$  be the text length, the index of Cole et al. in [5] solves the problem in time  $\mathcal{O}((\log n)^k \log \log n + m + occ)$ , but has the disadvantage of requiring  $\mathcal{O}(n(\log n)^{k+1})$  bits of space, often too much to be of practical interest. Better space requirements have been obtained at the price of search slow-down, with the solutions of Chan et al. in [6], where the authors propose an index requiring  $\mathcal{O}(n \log n)$  bits of space ( $\mathcal{O}(n)$  words) and  $\mathcal{O}(m + occ + (c \log n)^{k(k+1)} \log \log n)$  query time or, alternatively,  $\mathcal{O}(n)$  bits of space and  $\mathcal{O}(m + occ + (c \log n)^{k(k+2)} \log \log n \log^\epsilon n)$  query time (where  $c$  is a constant and  $\epsilon > 0$ ). The above bounds concern worst-case analysis. If average-case analysis is used, several interesting results have been proposed which improve upon worst-case bounds. The metric index of Chávez et al. presented in [7] exploiting the fact that the employed distance defines a metric space, was the first to remove the exponential dependency on the number of errors. This solution requires  $\mathcal{O}(m^{1+\sqrt{2}+\epsilon}n)$  bits of space and has expected  $\mathcal{O}(m^{1+\sqrt{2}+\epsilon} + occ)$  query time. Maaß and Nowak in [8] propose an index requiring  $\mathcal{O}(n \log^k n)$  bits of space and  $\mathcal{O}(m + occ)$  average query time (yet assuming a constant number of errors). Finally, the index of Navarro and Baeza-Yates proposed in [9] requires  $\mathcal{O}(n \log n)$  bits of space and has  $\mathcal{O}(n^\lambda \log n)$  average retrieval time, where  $\lambda < 1$  if  $k < m(1 - e/\sqrt{\sigma})$  and  $e$  is the natural logarithm base.

When space is a concern in the design of the data structure, the sheer *size* of such classic indexes as hash tables [3,10], suffix trees [11] or suffix arrays [12], soon becomes prohibitive. Succinct, compressed, and self indexes (see [13] for an accurate survey on the topic) are powerful notions that address and solve most of these problems in an efficient and elegant way. A text index is called *succinct* if it requires  $n \log \sigma + o(n \log \sigma)$  bits of space [14], *compressed* if the space is proportional to that of the compressed text, and *self index* if the index is compressed and does not require the original text to be stored in memory [13].

Suffix trees and suffix arrays can be implicitly represented in succinct or compressed space with such techniques as the FM index. Our first contribution shows that even hash indexes admit such a succinct representation. To obtain this result, we introduce a class of hash functions (namely, de Bruijn hash functions, that are homomorphisms on de Bruijn graphs) and use it to reduce space occupancy of hash-based text indexes from  $\Theta(n \log n)$  to  $n \log \sigma + o(n \log \sigma)$  bits, with only a  $\mathcal{O}(\log m)$  slow-down in the lookup operation.

We conclude illustrating the use of our proposed succinct hash data structure to describe a randomized algorithm for the  $k$ -mismatch problem operating in linear space and having smoothed complexity [15]  $\mathcal{O}((2\sigma)^k (\log n)^k (\log m + \xi) + (occ + 1) \cdot m)$ , where  $\xi = (mn)^{1+\log_2 c}$ ,  $c = (1 + (1 - 2\epsilon)^{m/\log_\sigma(mn)})/2$ ,

is a term which depends on  $m$ ,  $n$ , as well as on the amplitude  $\epsilon$  of the noise perturbing text and pattern. Smoothed analysis [15, 16] is a novel tool which interpolates continuously—through the parameter  $\epsilon$ —between worst and average case analysis, and therefore represents a more powerful tool than standard average case analysis which is often used to analyse the performances of randomized algorithms. Most importantly, we show that for *any*  $\epsilon > 0$ , if  $m$  is large enough, then  $\xi \in \mathcal{O}(\log m)$  and can, consequently, be ignored in the asymptotic analysis. Alternatively, to make our bound comparable with average-case results present in literature, one can set  $\epsilon = 0.5$  and obtain the standard average-case complexity  $\mathcal{O}((2\sigma)^k (\log n)^k \log m + (occ + 1) \cdot m)$ .

Our solution shows that, introducing randomization, it is possible to improve upon previous known upper bounds for the  $k$ -mismatch problem in linear space.

Our data structure has been implemented (among other tools) in the C++ *BWTIL* library (<https://github.com/nicolaprezza/BWTIL>) and has been integrated in the short-string alignment package ERNE, to be used in DNA analysis and freely downloadable at: <http://erne.sourceforge.net>

## 2 Notation

Throughout this paper we will work with the alphabet  $\Sigma = \{0, \dots, \sigma - 1\}$ ,  $\sigma = 2^r$ ,  $r > 0$ , and with hash functions of the form  $h : \Sigma^m \rightarrow \Sigma^w$  mapping length- $m$   $\Sigma$ -strings to length- $w$   $\Sigma$ -strings, where  $m \geq w$  and  $wr$  is considered the size of the memory-word (i.e. we assume that  $\sigma^w - 1$  fits into the computer memory-word). If necessary, we will use the symbol  ${}_w^m h$  instead of  $h$  when we need to be clear on  $h$ 's domain and codomain sizes. Given a string  $P \in \Sigma^m$ , the value  $h(P) \in \Sigma^w$  will be also dubbed the *fingerprint* of  $P$  (in  $\Sigma^w$ ). With  $T \in \Sigma^n$  we will denote the *text* that we want to index using our data structure.  $T_i^j$  will denote  $T[i, \dots, i + j - 1]$ , i.e. the  $j$ -th prefix of the  $i$ -th suffix of  $T$ . A hash data structure  $H$  for the text  $T$  with hash function  $h$ , will be a set of ordered pairs (an index) such that  $H = \{\langle h(T_i^m), i \rangle : 0 \leq i \leq n - m\}$ , that can be used to store and retrieve the positions of length- $m$  substrings of  $T$  ( $m$  is therefore fixed once the index is built). A *lookup* operation on the hash  $H$  given the fingerprint  $h(P)$ , will consist in the retrieval of all the positions  $0 \leq i < n$  such that  $\langle h(P), i \rangle \in H$  and cases where  $\langle h(P), i \rangle \in H$  but  $T_i^m \neq P$  will be referred to as *false positives*.

$\oplus$  is the exclusive OR (XOR) bitwise operator.  $a \oplus b$ , where  $a, b \in \Sigma$ , will indicate the bitwise XOR between (the bit representations of)  $a$  and  $b$  and, analogously,  $x \oplus y$ , where  $x, y \in \Sigma^m$  will indicate the bitwise XOR between (the bit representations of) the two words  $x$  and  $y$ .  $d_H(x, y)$  is the Hamming distance between  $x, y \in \Sigma^m$ .  $\Pr(E)$  is the probability of the event  $E$ .  $Be(p)$  is the bernoullian distribution with success probability  $p$ . If  $X$  is a random variable and  $f(X)$  a function of  $X$ , following [16],  $\mathbf{E}[f(X)]$  is the expected value of the random variable  $f(X)$ . If  $f(X) = X$  we simply write  $\mathbf{E}[X]$ . Logarithms are base 2 if not differently specified.

### 3 de Bruijn Functions and the dB-Hash Data Structure

In this section we present a technique that can be used to represent succinctly a hash index, using homomorphisms on de Bruijn graphs as hash functions. As we said in the introduction, even though hash indexes offer fast access times, their space requirements are usually quite high ( $\Theta(n \log n)$  bits). Our proposed solution consists in “compacting” the fingerprints in a text of size  $n - m + w$  that can then be indexed in succinct space using any of the popular techniques available in the literature. Central in our proposal is the introduction of a class of hash functions whose values on the text  $m$ -substrings *overlap*.

**Definition 1.** Let  $\Sigma = \{0, \dots, \sigma - 1\}$ . We say that a function  $h : \Sigma^m \rightarrow \Sigma^w$  is a de Bruijn hash function if and only if, for every pair of strings  $\sigma_1, \sigma_2 \in \Sigma^m$

$$\sigma_1[1, \dots, m - 1] = \sigma_2[0, \dots, m - 2] \Rightarrow h(\sigma_1)[1, \dots, w - 1] = h(\sigma_2)[0, \dots, w - 2].$$

This property guarantees that if two strings differ only for one character shift, then this happens also for their hash values. With the following definition we exhibit a de Bruijn hash function that will play an important role in the rest of our work:

**Definition 2.** Let  $\Sigma = \{0, \dots, 2^r - 1\}$ ,  $r > 0$ ,  $P \in \Sigma^m$ . With  $h_{\oplus} : \Sigma^m \rightarrow \Sigma^w$ ,  $w \leq m$  we denote the hash function defined as

$$h_{\oplus}(P) = \left( \bigoplus_{i=0}^{\lceil m/w \rceil - 2} P_{iw}^w \right) \oplus P_{m-w}^w$$

**Theorem 1.**  $h_{\oplus}$  is a de-Bruijn hash function.

*Proof.* The key observation is that a character shift in  $P$  produces a shift in each of the  $w$ -blocks XOR-ed by  $h_{\oplus}$ .

It is easy to show that de Bruijn hash functions correspond to homomorphisms on de Bruijn graphs (with set of nodes  $\Sigma^m$  and  $\Sigma^w$ ): intuitively, let  $G_m$  and  $G_w$  be two de Bruijn graphs with set of nodes  $\Sigma^m$  and  $\Sigma^w$ , respectively. Two nodes  $x, y \in \Sigma^m$  share an edge if and only if  $x[1, \dots, m - 1] = y[0, \dots, m - 2]$  (similarly for  $G_w$ ). Then, applying a de Bruijn hash function  ${}^m_w h$  to  $x$  and  $y$ , we obtain  $h(x), h(y) \in \Sigma^w$  such that (by Definition 1)  $h(x)[1, \dots, w - 1] = h(y)[0, \dots, w - 2]$ , i.e.  $h(x)$  and  $h(y)$  share an edge in  $G_w$ .

Given a de Bruijn hash function  ${}^m_w h : \Sigma^m \rightarrow \Sigma^w$  we can naturally “extend” it to another de Bruijn hash function  ${}^n_{n-m+w} h : \Sigma^n \rightarrow \Sigma^{n-m+w}$  operating on input strings of length  $n \geq m$  as described in the following definition.

**Definition 3.** Given  ${}^m_w h : \Sigma^m \rightarrow \Sigma^w$  de Bruijn hash function and  $n \geq m$ , the hash value of  ${}^n_{n-m+w} h$  on  $T \in \Sigma^n$ , is the unique string  ${}^n_{n-m+w} h(T) \in \Sigma^{n-m+w}$  such that:

$${}^n_{n-m+w} h(T)[i, \dots, i + w - 1] = {}^m_w h(T[i, \dots, i + m - 1]),$$

for every  $0 \leq i \leq n - m$ .

Since  ${}^m_w h$  univocally determines  ${}_{n-m+w}^n h$  and the two functions coincide on the common part  $\Sigma^m$  of their domain, in what follows we will simply use the symbol  $h$  to indicate both. Notice that the hash value  $h(T)$  can be trivially built in  $\mathcal{O}(mn/w)$  time exploiting Definition 3. However, particular hash functions may permit more efficient algorithms: the hash function  $h_{\oplus}$  defined in Definition 2, in particular, can be shown to have an optimal  $\mathcal{O}(n)$  time algorithm of this kind.

From Definitions 1 and 3 we can immediately derive the following important property:

**Lemma 1.** *If  $h$  is a de Bruijn hash function,  $n \geq m$ , and  $P \in \Sigma^m$  occurs in  $T \in \Sigma^n$  at position  $i$ , then  $h(P)$  occurs in  $h(T)$  at position  $i$ . The opposite implication does not (always) hold and we will refer to such cases as false positives.*

On the ground of Lemma 1 we can propose, differently from other approaches, to build a succinct index over the *hash value* of the text, instead of building it over the text. A crucial aspect is that this can be done while preserving our ability to locate substrings in the text, by simply turning our task into locating *fingerprints* in the hash of the text. We name our data structure a dB-hash.

### 3.1 Implementing the dB-Hash Data Structure

In order to create our dB-hash data structure, we build a succinct ( $n \log \sigma + o(n \log \sigma)$  bits) index over the hash  $h(T)$  of the text, augmenting it with further (light) structures described below. The problem of building a succinct—or compressed—index of a text has been extensively discussed in literature (see for example [17–19]), so here we omit the unnecessary details. Notice that, for reasons discussed in detail in section 4.3,  $h(T)$  could be very hard to compress. For this reason, here we present an uncompressed (yet succinct) version of our index. Briefly, our structure is an uncompressed FM index based on wavelet trees (similar to the one proposed in [19]). In our structure, suffix array pointers are sampled every  $\nu = \log^{1+\eta} n / \log \sigma$  positions of  $h(T)$ ,  $\eta > 0$  (as described in [17]) to reach  $o(n \log \sigma)$  bits of space and  $\mathcal{O}(\nu \log \sigma) = \mathcal{O}(\log^{1+\eta} n)$  time for the location of a pattern occurrence. This index supports search of a fingerprint  $f \in \Sigma^w$  in  $\mathcal{O}(w \log \sigma)$  time ( $w$  backward search steps, each of cost  $\log \sigma$ ).

With a *lookup* on the dB-hash we indicate the operation of retrieving the interval in  $h(T)^{BWT}$  corresponding to occurrences of the searched fingerprint. Since  $\mathcal{O}(w \log \sigma)$  cost for the lookup operation is far from the  $\mathcal{O}(1)$  cost guaranteed by a standard hash, we choose to speed-up this operation augmenting the structure with an auxiliary hash having overall memory occupancy of  $n / \log n$  bits. The auxiliary hash is used to record the results of the backward search algorithm (intervals on the BWT) on all the strings of length  $w_{aux} \leq w$ . A lookup operation on the dB-hash is then implemented with an initial lookup—on the auxiliary hash—of the  $w_{aux}$ -length suffix of the pattern’s fingerprint (cost  $\mathcal{O}(1)$ ) followed by backward search on the remaining portion of  $h(P)$  (cost  $\mathcal{O}((w - w_{aux}) \log \sigma)$ ). The  $n / \log n$  bits constraint on the auxiliary hash size limits  $w_{aux}$  to be  $w_{aux} = \log_{\sigma} n - 2 \log_{\sigma} \log n$ , so a lookup operation on our index requires  $\mathcal{O}(w - \log_{\sigma} n + 2 \log_{\sigma} \log n)$  backward search steps. In section 4.3 we will

show that this auxiliary structure asymptotically reduces the cost of a lookup operation from  $\mathcal{O}(\log n)$  to  $\mathcal{O}(\log m)$ .

Summing up, the dB-hash data structure is constituted by a succinct index over  $h(T)$  augmented with an auxiliary hash of size  $n/\log n$  bits. This amounts to an overall space occupancy of  $n \log \sigma + o(n \log \sigma)$  bits: the index<sup>1</sup> is succinct.

## 4 de Bruijn Hash for the $k$ -Mismatch Problem

The  $k$ -mismatch problem asks to find all occurrences up to  $k$  errors (under the Hamming distance) of a given pattern  $P$  in a given text  $T$ . In this section we use the results of [3, 4] and Section 3 to describe an algorithm for this problem having low smoothed complexity while requiring only linear space for the index.

### 4.1 Squeezing the Search Space: Hamming-Aware Functions

The core of our searching procedure is based on the algorithm *rNA* (Vezi et al. [3], Policriti et al. [4]), a hash-based randomized numerical aligner based on the concept of *Hamming-aware* hash functions. Hamming-aware hash functions are particular hash functions designed to “squeeze” the  $k$ -radius Hamming ball centered on a pattern  $P$ , to a  $\mathcal{O}(k)$ -radius Hamming ball centered on the hash value  $h(P)$  of  $P$ . This feature allows to search much more efficiently, reducing search space size from  $\mathcal{O}(m^k)$  to  $\mathcal{O}(w^k) = \mathcal{O}((\log n)^k)$ . More formally:

**Definition 4.** *A hash function  $h$  is Hamming-aware if there exist*

- a set  $\mathcal{Z}(k) \subseteq \Sigma^w$  such that  $|\mathcal{Z}(k)| \in \mathcal{O}(c^k w^k)$ , for some constant  $c$ , and
- a binary operation  $\phi : \Sigma^w \times \Sigma^w \rightarrow \Sigma^w$  computable in  $\mathcal{O}(w)$  time,

such that if  $P \in \Sigma^m$  then the following inclusion holds:

$$\{h(P') : P' \in \Sigma^m, d_H(P, P') \leq k\} \subseteq \{h(P) \phi z : z \in \mathcal{Z}(k)\} \quad (1)$$

Given a query  $P$ , the algorithm *rNA* computes its fingerprint and efficiently retrieves all the fingerprints of strings  $P'$  such that  $d_H(P, P') \leq k$ . This is done computing  $h(P) \phi z$ , for  $z \in \mathcal{Z}(k)$  and searching the index for each one of them.

Our search algorithm will have an overall structure that remains essentially the same described in [3] and [4]. In order to couple the *rNA* technique with the use of our proposed dB-hash, we only need to prove the existence of de Bruijn hash functions satisfying the Hamming awareness condition. The following theorem shows that our exclusive-or based function is a possible solution:

**Theorem 2.** *The de Bruijn hash function  $h_{\oplus}$  defined in Definition 2 is a Hamming-aware hash function. In particular:*

<sup>1</sup> Notice that this space is required to store the succinct hash *index* only; checking for false positives requires also the storage of the text, for  $n \log \sigma$  bits of additional space consumption.



- The binary operation  $\phi$  for  $h_{\oplus}$  is  $\oplus$ .
- $\mathcal{Z}(k) = \{h_{\oplus}(P_1) \oplus h_{\oplus}(P_2) : d_H(P_1, P_2) \leq k, P_1, P_2 \in \Sigma^m\}$  has  $\mathcal{O}((2\sigma w)^k)$  elements.

*Proof.* First, it can be proved that representing  $h_{\oplus}$  by a matrix (notice that  $h_{\oplus}$  is a linear map), it has at most  $2w - 1$  distinct columns. The claim then follows from the fact that the elements of  $\mathcal{Z}(k)$  can be built XOR-ing together at most  $k$  columns of  $h_{\oplus}$ .

$\mathcal{Z}(k)$  needs not to be explicitly stored in memory. Instead, we can compute each  $z \in \mathcal{Z}(k)$  in  $\mathcal{O}(1)$  time on-the-fly during search, exploiting the following tree representation  $\mathcal{T}(k)$  of  $\mathcal{Z}(k)$ . Let  $v$  be a node of  $\mathcal{T}(k)$  with  $l(v) \in \Sigma^w$  the label of  $v$ . The root  $r$  is such that  $l(r) = 0^w$ . Each node  $v$  in  $\mathcal{T}(k)$  has  $|\mathcal{Z}(1)|$  children  $v(0), \dots, v(|\mathcal{Z}(1)| - 1)$ , where  $l(v(i)) = l(v) \oplus \mathcal{Z}(1)[i]$  ( $\mathcal{Z}(j)[i]$  being the  $i$ -th element in the set  $\mathcal{Z}(j)$ ). The height of  $\mathcal{T}(k)$  is  $k$ : as a consequence, its size (number of nodes) is  $\mathcal{O}((2\sigma w)^k)$ . It can be shown that, if  $\text{depth}(v) = i$  then  $l(v) \in \mathcal{Z}(i)$ . Conversely, if  $z \in \mathcal{Z}(i)$  then there is at least one node  $v$  of  $\mathcal{T}(k)$  such that  $l(v) = z$  and  $\text{depth}(v) \leq i$ .  $\mathcal{T}(k)$  can be dfs-visited during the search memorizing only the set  $\mathcal{Z}(1)$  ( $\mathcal{O}(\sigma w \log \sigma)$  bits) and, for each node  $v$  in the current path, its label  $l(v)$  ( $w \log \sigma$  bits) and a counter on the elements of  $\mathcal{Z}(1)$ . This representation does not penalize performances and has a total space consumption of  $\mathcal{O}(kw \log \sigma + \sigma w \log \sigma)$  bits.

## 4.2 The dB-rNA Algorithm

Let us briefly describe and analyse our algorithm, putting together the results presented throughout the paper to tackle the  $k$ -mismatch problem. We name our algorithm dB-rNA (de Bruijn randomized numerical aligner).

Given a pattern  $P$ , the algorithm computes its fingerprint  $h_{\oplus}(P)$  ( $\mathcal{O}(m)$  steps) and, for each element  $z$  in  $\mathcal{Z}(k)$ , it executes a lookup in position  $h_{\oplus}(P) \oplus z$  of the dB-hash data structure ( $\mathcal{O}(w - \log_{\sigma} n + 2 \log_{\sigma} \log n)$  steps for each lookup). Each lookup is followed by some BWT-to-text coordinate conversions ( $\mathcal{O}(\log^{1+\eta} n)$  time for each entry in position  $h_{\oplus}(P) \oplus z$  of the dB-hash). For each text coordinate  $i$  obtained in the previous step, the algorithm compares then the pattern with the text substring  $T_i^m$  to detect false positives ( $\mathcal{O}(m)$  for each text position). The space required for the execution is that of the dB-hash data structure ( $n \log \sigma + o(n \log \sigma)$  bits) plus that of the plain text ( $n \log \sigma$  bits).

## 4.3 Complexity Analysis of the Algorithm

In order to study the complexity of our algorithm we need to establish an upper bound on the expected collision lists length in the hash table. To accomplish this task we chose to use smoothed analysis of D. A. Spielman and S. Teng [15], a tool that has already been used in previous works for the analysis of string matching algorithms—see [20]. Smoothed analysis aims at explaining the behavior of algorithms in practice, where often standard worst and average case analysis do not

provide meaningful bounds (see, for example, [16]). The key observation motivating smoothed analysis is that, in practice, data is firstly generated by a source and then perturbed by random noise (for example channel noise in communication theory or random genetic mutations in bioinformatics). While studying complexity, the former step translates in the choice of a worst-case instance (as in worst-case analysis) and the latter in the computation of an expected complexity, with the noise being the source of randomness (as in average-case analysis). As a by-product, smoothed analysis does not require to make assumptions—often hard to motivate—on the distribution of input data.

In this section we first use smoothed analysis theory to give a (rather general) result relating the presence of random noise perturbing text and pattern to the expected collision lists length in the hash table. Our results are general and apply to any hash function with the property of being a linear map between  $\Sigma^m$  and  $\Sigma^w$  seen as vector spaces. Our bounds are then used to compute the expected hash load distribution induced by  $h_{\oplus}$  and, consequently, the smoothed complexity of our algorithm.

**Smoothed Analysis of Hashing with Linear Maps.** Here we focus on hash functions  $h$  that are linear maps between  $\Sigma^m$  and  $\Sigma^w$  seen as vector spaces. We use the same symbol  $h$  to indicate also the characteristic matrix  $h \in \Sigma^{w \times m}$  associated with the linear map  $h$ ; the specific interpretation of the symbol  $h$  will be clear from the context. Even though our results could be stated in full generality with respect to  $\Sigma = \mathbb{Z}_{\sigma}$  and sum modulo  $\sigma$ , for simplicity, we will give them for the case  $\Sigma = \mathbb{Z}_2$  with the corresponding sum operator being  $\oplus$ . One of the advantages of this choice is that, in practice, fingerprints can be manipulated in time  $\mathcal{O}(1)$ , since most of the modern computer architectures provide bitwise XOR operator implemented in hardware. Let

$$O_h(T, P) = |\{i : h(T_i^m) = h(P), 0 \leq i \leq n - m\}|,$$

be the number of text substrings of length  $m$  mapped by  $h$  to the value  $h(P)$ , i.e. the length of the collision-list in position  $h(P)$ .

In smoothed analysis, usually, the (whole) problem instance is considered to be perturbed. In our case an instance is the query pair  $\langle T, P \rangle$  constituted by the text  $T \in \Sigma^n$  (to be indexed) and the pattern  $P \in \Sigma^m$  (to be searched in  $T$ ). Let  $\tau \in \Sigma^n$  and  $\pi \in \Sigma^m$  be the random noise vectors perturbing the text and the pattern, respectively.  $\tilde{T} = T \oplus \tau$  and  $\tilde{P} = P \oplus \pi$  are the perturbed text and the perturbed pattern, respectively.  $\tau_i$  and  $\pi_j$ ,  $0 \leq i < n$ ,  $0 \leq j < m$ , are independent and identically distributed as  $Be(\epsilon)$ , with  $0 \leq \epsilon \leq 0.5$ . Adopting a notation similar to the one introduced in [15, 16], we define the smoothed hash-load distribution induced by  $h : \Sigma^m \rightarrow \Sigma^w$  on texts of length  $n$  to be

$$\text{Smoothed}_h^{\epsilon}(n) = \max_{T \in \Sigma^n, P \in \Sigma^m} \mathbf{E}_{\tau, \pi} [ O_h(\tilde{T}, \tilde{P}) ] \quad (2)$$

We will first work with linear maps  $h : \Sigma^m \rightarrow \Sigma^w$  such that  $\sum_{i=0}^{w-1} h_{ij} \leq 1$  for all  $j = 0, \dots, m - 1$ , and define  $t_h = \min_{i=0, \dots, w-1} \left\{ \sum_{j=0}^{m-1} h_{ij} \right\}$  and  $c_h = (1 + (1 - 2\epsilon)^{t_h}) / 2$ . The following theorem holds.

**Theorem 3.**  $\text{Smoothed}_h^\epsilon(n) \leq n(c_h)^w$ .

*Proof.* The proof is based on an analysis of the  $\oplus$  of bernoullian r.v.'s, using the hypotheses on the number of 1's on rows and columns.

A weaker bound can be obtained on a a more general linear map  $h$  by a simple transformation.

**Corollary 1.** *Given  $h : \Sigma^m \rightarrow \Sigma^w$ , let  $h'$  be the linear map defined as*

$$h'_{ij} = \begin{cases} h_{ij} & \text{if } \sum_{i=0}^{w-1} h_{ij} \leq 1 \\ 0 & \text{otherwise} \end{cases}, \quad 0 \leq i < w, 0 \leq j < m.$$

*Then,  $\text{Smoothed}_h^\epsilon(n) \leq n(c_{h'})^w$ .*

Since  $c_{h'}$  and  $t_{h'}$  are univocally determined from  $h$ , when clear from the context we will denote them simply by  $c$  and  $t$ .

In case of a more general alphabet  $\Sigma = \{0, \dots, 2^r - 1\}$ , simply considering a  $\Sigma$ -digit as a group of  $r > 0$  consecutive bits and  $h$  as a matrix of size  $wr \times mr$  with elements in  $\mathbb{Z}_2$ , the previous bound becomes:

**Corollary 2.**  $\text{Smoothed}_h^\epsilon(n) \leq n2^{rw \log_2 c} = n\sigma^{w \log_2 c}$ .

As expected this bound interpolates, through the parameter  $\epsilon$ , between worst-case and average-case analysis: if  $\epsilon = 0$  (absence of noise) then  $\sigma^{w \log_2 c} = 1$  and we obtain the worst-case analysis  $\text{Smoothed}_h^\epsilon(n) \leq n$ . If  $\epsilon = 0.5$  (uniform noise) then  $\sigma^{w \log_2 c} = \sigma^{-w}$  and we obtain  $\text{Smoothed}_h^\epsilon(n) \leq n\sigma^{-w}$ : the expected hash load induced by a uniform random text (as predicted by average-case analysis).

**Analysis of the Algorithm.** The smoothed complexity of the dB-rNA algorithm is defined as (see [16]):

$$\text{Smoothed}_{\text{dB-rNA}}^\epsilon(n, m) = \max_{T \in \Sigma^n, P \in \Sigma^m} \mathbf{E}_{\tau, \pi} [ \mathcal{T}_{\text{dB-rNA}}(\langle \tilde{T}, \tilde{P} \rangle) ] \quad (3)$$

Using Corollary 2 and the definition of  $h_\oplus$  the following lemma can be proved.

**Lemma 2.**  $\text{Smoothed}_{h_\oplus}^\epsilon(n) \in \mathcal{O}(n\sigma^{w \log_2 c})$ , where  $c = (1 + (1 - 2\epsilon)^{m/w}) / 2$ .

*Proof.* First prove that each bit of  $h_\oplus(x)$  is the XOR of  $\mathcal{O}(m/w)$  bits of  $x$ . The result then follows from Corollary 2.

Letting  $\nu = \log^{1+\eta} n$  be the cost of a BWT-to-text coordinate conversion (see section 3.1), we will firstly make our calculations assuming  $m \geq \nu$  (so that the coordinate conversion cost is absorbed by the cost of checking for a false positive). This assumption simplifies the notation and is quite reasonable

(considering that the main theoretic interest is on large patterns); however, for completeness at the end of this section we will also give time bounds for patterns such that  $m < \nu$ .

According to Lemma 2, Theorem 2, and the algorithm description in section 4.2, (3) has the following upper bound:

$$\mathcal{O}(|\mathcal{Z}(k)| \cdot ((w - \log_\sigma n + 2 \log_\sigma \log n) \log \sigma + mn\sigma^{w \log_2 c}) + (occ + 1) \cdot m) \tag{4}$$

where  $|\mathcal{Z}(k)| \in \mathcal{O}((2\sigma w)^k)$  and  $c = (1 + (1 - 2\epsilon)^{m/w})/2$ .

At this point we can determine the optimal word size  $w_{opt}$  for which the above complexity reaches its minimum. To simplify the analysis we assume an uniform distribution for the hash load, i.e.  $\log_2 c = -1$  (this choice will affect only the value found for  $w_{opt}$ ). Intuitively, decreasing  $w$  the term  $\mathcal{O}(n\sigma^{-w})$  increases exponentially while increasing it the term  $|\mathcal{Z}(k)|$  increases polynomially, so it is reasonable that (4) has a unique minimum  $w_{opt}$ . Let  $\mathcal{C}(n, m, w, k, \sigma)$  be the complexity (4) where  $\log_2 c = -1$ . Solving  $\partial\mathcal{C}(n, m, w_{opt}, k, \sigma)/\partial w_{opt} = 0$  one can obtain

$$w_{opt} = \log_\sigma(mn) + \log_\sigma \left( \frac{1 - k/w_{opt}}{\log \sigma \cdot ((k + 1) - k/w_{opt} \cdot (\log n - \log \log n))} \right) \tag{5}$$

We assume that  $k \approx 0$ , so the second term in (5) is small and can be ignored. Notice that, using  $w_{opt} = \log_\sigma(mn)$  as word size, a lookup operation in the dB-hash requires  $\mathcal{O}(w_{opt} - \log_\sigma n + 2 \log_\sigma \log n) = \mathcal{O}(\log m / \log \sigma)$  backward search steps ( $\mathcal{O}(\log m)$  time). Substituting  $w_{opt} = \log_\sigma(mn)$  in (4) we finally obtain:

**Theorem 4.** *The smoothed complexity of the dB-rNA algorithm is*

$$\mathcal{O}((2\sigma)^k (\log n)^k (\log m + \xi) + (occ + 1) \cdot m)$$

where  $\xi = (mn)^{1+\log_2 c}$  and  $c = (1 + (1 - 2\epsilon)^{m/\log_\sigma(mn)})/2$ .

Alternatively, to make our bound comparable with average-case results present in literature, one can set  $\epsilon = 0.5$  and obtain:

**Theorem 5.** *The expected complexity (on uniformly distributed inputs) of the dB-rNA algorithm is*

$$\mathcal{O}((2\sigma)^k (\log n)^k \log m + (occ + 1) \cdot m)$$

We point out that for any  $\epsilon > 0$  and  $m$  large enough, the term  $\xi$  in Theorem 4 is small ( $\xi \leq \log m$ ) and can be ignored<sup>2</sup>. This is a strong result since it does not make restrictive assumptions on the amplitude  $\epsilon$  of the noise perturbing the input instance and shows that, asymptotically, the smoothed and the expected complexities of our algorithm coincide.

As stated above, these time bounds hold only for patterns such that  $m \geq \nu$ . For short patterns such that  $m < \nu$  the cost of a coordinate conversion dominates that of the false-positive check, and our bound becomes  $\mathcal{O}((2\sigma)^k (\log n)^k (\log m + \xi) + (occ + 1) \cdot \log^{1+\eta} n)$ .

<sup>2</sup> Such minimum  $m$  satisfies the inequality  $(mn)^{1+\log_2 c} \leq \log m$ .

## 5 Conclusions and Final Remarks

In this work we tackled one of the main bottlenecks of hashing, that is its space requirements, introducing a strategy integrating hashing and (succinct) indexing. We presented a succinct index, called dB-hash, designed on the hash value  $h(T)$  of the text. This is done using homomorphisms on de Bruijn graphs as hash functions—here dubbed de Bruijn hash functions. We proved that de Bruijn hash functions with the additional feature of being Hamming aware—a property granting the ability to significantly reduce search space—exist: our algorithm improves upon previous linear-space strategies discussed in literature, and is one of the few results taking into account randomization to solve the  $k$ -mismatch problem. Moreover, we presented a smoothed analysis of hashing, i.e. the use of smoothed analysis theory in the study of load distribution in a hash table.

As every hash-based index, our index suffers from the limitation that pattern length  $m$  is fixed and need to be known at index construction time. This is a problem shared with others indexes for approximate pattern matching (see for example [7]), and with hash indexes in general. A second drawback is the increased query time with respect to the standard hash version ( $\mathcal{O}(\log m)$  for lookup and  $\mathcal{O}(\log^{1+\eta} n)$  for coordinate conversions). Despite this fact, we point out that in practical implementations on large texts, only the dB-hash data structure is able to use the optimal word size  $w_{opt} = \log_{\sigma}(mn)$ , and can thus reach the optimal query time. As an example, consider indexing the Human genome ( $n = 3.2 \times 10^9$ ,  $\sigma = 4$ ) with pattern length  $m = 30$ . The optimal word size is  $w_{opt} = 19$ . While the dB-hash data structure still requires  $\mathcal{O}(n \log \sigma)$  bits of space, a standard hash would require  $\sigma^{w_{opt}} \log n$  bits  $\approx 1 TB$  of space *only* for the lookup table, which is clearly unacceptable in practice. As a result, standard hash tables are limited to sub-optimal word sizes and thus to sub-optimal query times.


We think this paper opens a number of possibilities for future work. First of all notice that is possible a generalization of our results to the smoothed analysis of hashing with alphabets of general size  $\sigma$  (this can be done considering linear maps modulo  $\sigma$  as hash functions). Then observe that the  $h(T)$  construction technique can be used as a text-transformation preprocessing, randomizing  $T$ , and coupled with existing pattern matching algorithms. Finally, combining our strategy with metric indexes, such as the one presented in [7], or through an extension of the notion of Hamming-awareness to a more general distance-awareness, could lead to a generalization of our results to more complex distances—such as the edit distance.

## References

1. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.L., et al.: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.* **10**(3), R25 (2009)
2. Li, R., Yu, C., Li, Y., Lam, T.W., Yiu, S.M., Kristiansen, K., Wang, J.: SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics* **25**(15), 1966–1967 (2009)

3. Vezzi, F., Del Fabbro, C., Tomescu, A.I., Policriti, A.: rNA: a fast and accurate short reads numerical aligner. *Bioinformatics* **28**(1), 123–124 (2012)
4. Policriti, A., Tomescu, A.I., Vezzi, F.: A randomized numerical aligner (rna). *J. Comput. Syst. Sci.* **78**(6), 1868–1882 (2012)
5. Cole, R., Gottlieb, L.A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, pp. 91–100. ACM (2004)
6. Chan, H.-L., Lam, T.-W., Sung, W.-K., Tam, S.-L., Wong, S.-S.: A linear size index for approximate pattern matching. In: Lewenstein, M., Valiente, G. (eds.) *CPM 2006*. LNCS, vol. 4009, pp. 49–59. Springer, Heidelberg (2006)
7. Chávez, E., Navarro, G.: A metric index for approximate string matching. In: Rajsbaum, S. (ed.) *LATIN 2002*. LNCS, vol. 2286, p. 181. Springer, Heidelberg (2002)
8. Maaß, M.G., Nowak, J.: Text indexing with errors. *Journal of Discrete Algorithms* **5**(4), 662–681 (2007)
9. Navarro, G., Baeza-Yates, R.: A hybrid indexing method for approximate string matching. *Journal of Discrete Algorithms* **1**(1), 205–239 (2000)
10. Li, R., Li, Y., Kristiansen, K., Wang, J.: SOAP: short oligonucleotide alignment program. *Bioinformatics* **24**(5), 713–714 (2008)
11. Weiner, P.: Linear pattern matching algorithms. In: *IEEE Conference Record of 14th Annual Symposium on Switching and Automata Theory, SWAT 2008*, pp. 1–11. IEEE (1973)
12. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing* **22**(5), 935–948 (1993)
13. Navarro, G., Mäkinen, V.: Compressed full-text indexes. *ACM Computing Surveys (CSUR)* **39**(1), 2 (2007)
14. Jacobson, G.: Space-efficient static trees and graphs. In: *30th Annual Symposium on Foundations of Computer Science*, pp. 549–554. IEEE (1989)
15. Spielman, D.A., Teng, S.H.: Smoothed analysis: an attempt to explain the behavior of algorithms in practice. *Communications of the ACM* **52**(10), 76–84 (2009)
16. Spielman, D., Teng, S.H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In: *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pp. 296–305. ACM (2001)
17. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pp. 390–398. IEEE (2000)
18. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 841–850. Society for Industrial and Applied Mathematics (2003)
19. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: An alphabet-friendly FM-index. In: Apostolico, A., Melucci, M. (eds.) *SPIRE 2004*. LNCS, vol. 3246, pp. 150–160. Springer, Heidelberg (2004)
20. Andoni, A., Krauthgamer, R.: The smoothed complexity of edit distance. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I*. LNCS, vol. 5125, pp. 357–369. Springer, Heidelberg (2008)

# Top- $k$ Term-Proximity in Succinct Space

J. Ian Munro<sup>1</sup>, Gonzalo Navarro<sup>2</sup>, Jesper Sindhil Nielsen<sup>3</sup>, Rahul Shah<sup>4</sup>,  
and Sharma V. Thankachan<sup>5</sup> 

<sup>1</sup> Cheriton School of CS, University of Waterloo, Waterloo, Canada  
`imunro@uwaterloo.ca`

<sup>2</sup> Department of CS, University of Chile, Santiago, Chile  
`gnavarro@dcc.uchile.cl`

<sup>3</sup> MADALGO, Aarhus University, Aarhus, Denmark  
`jasn@cs.au.dk`

<sup>4</sup> School of EECS, Louisiana State University, Louisiana, USA  
`rahul@csc.lsu.edu`

<sup>5</sup> School of CSE, Georgia Institute of Technology, Georgia, USA  
`sharma.thankachan@gmail.com`

**Abstract.** Let  $\mathcal{D} = \{\mathbb{T}_1, \mathbb{T}_2, \dots, \mathbb{T}_D\}$  be a collection of  $D$  string documents of  $n$  characters in total, that are drawn from an alphabet set  $\Sigma = [\sigma]$ . The *top- $k$*  document retrieval problem is to preprocess  $\mathcal{D}$  into a data structure that, given a query  $(P[1..p], k)$ , can return the  $k$  documents of  $\mathcal{D}$  most relevant to pattern  $P$ . The relevance is captured using a predefined ranking function, which depends on the set of occurrences of  $P$  in  $\mathbb{T}_d$ . For example, it can be the term frequency (i.e., the number of occurrences of  $P$  in  $\mathbb{T}_d$ ), or it can be the term proximity (i.e., the distance between the closest pair of occurrences of  $P$  in  $\mathbb{T}_d$ ), or a pattern-independent importance score of  $\mathbb{T}_d$  such as PageRank. Linear space and optimal query time solutions already exist for this problem. Compressed and compact space solutions are also known, but only for a few ranking functions such as term frequency and importance. However, space efficient data structures for term proximity based retrieval have been evasive. In this paper we present the first sub-linear space data structure for this relevance function, which uses only  $o(n)$  bits on top of any compressed suffix array of  $\mathcal{D}$  and solves queries in time  $O((p+k) \text{ polylog } n)$ .

## 1 Introduction

Ranked document retrieval, that is, returning the documents that are most relevant to a query, is the fundamental task in Information Retrieval (IR) [1, 6]. Muthukrishnan [19] initiated the study of this family of problems in the more general scenario where both the documents and the queries are general strings over arbitrary alphabets, which has applications in several areas [20]. In this scenario, we have a collection  $\mathcal{D} = \{\mathbb{T}_1, \mathbb{T}_2, \dots, \mathbb{T}_D\}$  of  $D$  string documents of total length  $n$ , drawn from an alphabet  $\Sigma = [\sigma]$ , and the query is a pattern  $P[1..p]$

---

Funded in part by NSERC of Canada and the Canada Research Chairs program, Fondecyt Grant 1-140796, Chile, and NSF Grants CCF-1017623, CCF-1218904.

over  $\Sigma$ . Muthukrishnan considered a family of problems called *thresholded* document listing: given an additional parameter  $K$ , list only the documents where some function  $\text{score}(P, d)$  of the occurrences of  $P$  in  $T_d$  exceeded  $K$ . For example, the *document mining* problem aims to return the documents where  $P$  appears at least  $K$  times, whereas the *repeats* problem aims to return the documents where two occurrences of  $P$  appear at distance at most  $K$ . While document mining has obvious connections with typical term-frequency measures of relevance [1, 6], the repeats problem is more connected to various problems in bioinformatics [4, 12]. Also notice that the repeats problem is closely related to the term proximity based document retrieval in IR field [5, 29, 32–34]. Muthukrishnan achieved optimal time for both problems, with  $O(n)$  space (in words) if  $K$  is specified at indexing time and  $O(n \log n)$  if specified at query time.

A more natural version of the thresholded problems, as used in IR, is *top- $k$  retrieval*: Given  $P$  and  $k$ , return  $k$  documents with the best  $\text{score}(P, d)$  values. Hon et al. [15, 16] gave a general framework to solve top- $k$  problems for a wide variety of  $\text{score}(P, d)$  functions, which takes  $O(n)$  space, allows  $k$  to be specified at query time, and solves queries in  $O(p + k \log k)$  time. Navarro and Nekrich [22] reduced the time to  $O(p + k)$ , and finally Shah et al. [30] achieved time  $O(k)$  given the locus of  $P$  in the generalized suffix tree of  $\mathcal{D}$ . Recently, Munro et al. [18] introduced an  $O(n)$ -word index, that can find the top- $k$ th document in  $O(\log k)$  time, once the locus of  $P$  is given.

The problem is far from closed, however. Even the  $O(n)$  space (i.e.,  $O(n \log n)$  bits) is excessive compared to the size of the text collection itself ( $n \log \sigma$  bits), and in data-intensive scenarios it often renders all these solutions impractical by a wide margin. Hon et al. [16] also introduced a general framework for *succinct indexes*, which use  $o(n)$  bits<sup>1</sup> on top of a *compressed suffix array* (CSA) [21], which represents  $\mathcal{D}$  in a way that also provides pattern-matching functionalities on it, all within space ( $|\text{CSA}|$ ) close to that of the *compressed* collection. A CSA finds the suffix array interval of  $P[1..p]$  in time  $t_s(p)$  and retrieves any cell of the suffix array or its inverse in time  $t_{\text{SA}}$ . Hon et al. achieved  $O(t_s(p) + k t_{\text{SA}} \log^{3+\epsilon} n)$  query time, using  $O(n/\log^\epsilon n)$  bits. Subsequent work (see [20, 26]) improved the initial result up to  $O(t_s(p) + k t_{\text{SA}} \log^2 k \log^\epsilon n)$  [24], and also considered *compact indexes*, which may use  $o(n \log n)$  bits on top of the CSA. For example, these achieve  $O(t_s(p) + k t_{\text{SA}} \log k \log^\epsilon n)$  query time using  $n \log \sigma + o(n)$  further bits [14], or  $O(t_s(p) + k \log^* k)$  query time using  $n \log D + o(n \log n)$  further bits [25].

However, all these succinct and compact indexes work *exclusively* for the term frequency (or closely related, e.g., TF-IDF) measure of relevance. For the simpler case where documents have a fixed relevance independent of  $P$ , succinct indexes achieve  $O(t_s(p) + k t_{\text{SA}} \log k \log^\epsilon n)$  query time [3], and compact indexes using  $n \log D + o(n \log D)$  bits achieve  $O(t_s(p) + k \log(D/k))$  time [10]. On the other hand, there have been *no succinct nor compact indexes for the term proximity measure of relevance*,  $\text{tp}(P, d) = \min\{|i-j| > 0, T_d[i..i+p-1] = T_d[j..j+p-1] = P\} \cup \{\infty\}$ . In this paper we introduce the first such result as follows.

<sup>1</sup> If  $D = o(n)$ , which we assume for simplicity in this paper. Otherwise it is  $D \log(n/D) + O(D) + o(n)$  bits.



**Theorem 1.** *Using a CSA plus  $o(n)$  bits data structure, one can answer top- $k$  term proximity queries in  $O(t_s(p) + (\log^2 n + k(t_{\text{SA}} + \log k \log n)) \log^{2+\varepsilon} n)$  time, for any constant  $\varepsilon > 0$ .*

## 2 Basic Concepts

Let  $T[1..n] = T_1 \circ T_2 \circ \dots \circ T_D$  be the text (from an alphabet  $\Sigma = [\sigma] \cup \{\$\}$ ) obtained by concatenating all the documents in  $\mathcal{D}$ . Each document is terminated with a special symbol  $\$$ , which does not appear anywhere else. A suffix  $T[i..n]$  of  $T$  belongs to  $T_d$  iff  $i$  is in the region corresponding to  $T_d$  in  $T$ . Thus, it holds  $d = 1 + \text{rank}_B(i - 1)$ , where  $B[1..n]$  is a bitmap defined as  $B[j] = 1$  iff  $T[j] = \$$  and  $\text{rank}_B(i - 1)$  is the number of 1s in  $B[1..i - 1]$ . This operation is computed in  $O(1)$  time on a representation of  $B$  that uses  $D \log(n/D) + O(D) + o(n)$  bits [28]. For simplicity, we assume  $D = o(n)$ , and thus  $B$  uses  $o(n)$  bits.

*Suffix Tree* [31] of  $T$  is a compact trie containing all of its suffixes, where the  $i$ th leftmost leaf,  $\ell_i$ , represents the  $i$ th lexicographically smallest suffix. It is also called the generalized suffix tree of  $\mathcal{D}$ , GST. Each edge in GST is labeled by a string, and  $\text{path}(x)$  is the concatenation of the edge labels along the path from the GST root to node  $x$ . Then  $\text{path}(\ell_i)$  is the  $i$ th lexicographically smallest suffix of  $T$ . The highest node  $x$  with  $\text{path}(x)$  prefixed by  $P[1..p]$  is the *locus* of  $P$ , and is found in time  $O(p)$  from the GST root. The GST uses  $O(n)$  words of space.

*Suffix Array* [17] of  $T$ ,  $\text{SA}[1..n]$ , is defined as  $\text{SA}[i] = n + 1 - |\text{path}(\ell_i)|$ , the starting position in  $T$  of the  $i$ th lexicographically smallest suffix of  $T$ . The *suffix range* of  $P$  is the range  $\text{SA}[sp, ep]$  pointing to the suffixes that start with  $P$ ,  $T[\text{SA}[i].. \text{SA}[i] + p - 1] = P$  for all  $i \in [sp, ep]$ . Also,  $\ell_{sp}$  (resp.,  $\ell_{ep}$ ) are the leftmost (resp., rightmost) leaf in the subtree of the locus of  $P$ .

*Compressed Suffix Array* [8, 11, 21] of  $T$ , CSA, is a compressed representation of SA, and usually also of  $T$ . Its size in bits,  $|\text{CSA}|$ , is  $O(n \log \sigma)$  and usually much less. The CSA finds the interval  $[sp, ep]$  of  $P$  in time  $t_s(p)$ . It can output any value  $\text{SA}[i]$ , and even of its inverse permutation,  $\text{SA}^{-1}[i]$ , in time  $t_{\text{SA}}$ . For example, a CSA using  $nH_h(T) + o(n \log \sigma)$  bits [2] gives  $t_s(p) = O(p)$  and  $t_{\text{SA}} = O(\log^{1+\varepsilon} n)$  for any constant  $\varepsilon > 0$ , where  $H_h$  is the  $h$ th order empirical entropy.

*Compressed Suffix Tree* of  $T$ , CST, is a compressed representation of GST, where node identifiers are their corresponding suffix array ranges. The CST can use  $o(n)$  bits on top of a CSA [23] and compute (among others) the lowest common ancestor (LCA) of two leaves  $\ell_i$  and  $\ell_j$ , in time  $O(t_{\text{SA}} \log^\varepsilon n)$ , and the Weiner link  $\text{Wlink}(a, v)$ , which leads to the node with path label  $a \circ \text{path}(v)$ , in time  $O(t_{\text{SA}})$ .<sup>2</sup>

*Orthogonal Range Successor/Predecessor.* Given  $n$  points in  $[n] \times [n]$ , an  $O(n \log n)$ -bit data structure can retrieve the point in a given rectangle with lowest

<sup>2</sup> Using  $O(n / \log^\varepsilon n)$  bits and no special implementation for operations  $\text{SA}^{-1}[\text{SA}[i] \pm 1]$ .

$y$ -coordinate value, in time  $O(\log^\epsilon n)$  for any constant  $\epsilon > 0$  [27]. Combined with standard range tree partitioning, the following result easily follows.

**Lemma 1.** *Given  $n'$  points in  $[n] \times [n] \times [n]$ , a structure using  $O(n' \log^2 n)$  bits can support the following query in  $O(\log^{1+\epsilon} n)$  time, for any constant  $\epsilon > 0$ : find the point in a region  $[x, x'] \times [y, y'] \times [z, z']$  with the lowest/highest  $x$ -coordinate.*

### 3 An Overview of Our Data Structure

The top- $k$  term proximity is related to a problem called *range restricted searching*, where one must report all the occurrences of  $P$  that are within a text range  $\mathbb{T}[i..j]$ . It is known that succinct data structures for that problem are unlikely to exist in general, whereas indexes of size  $|\text{CSA}| + O(n/\log^\epsilon n)$  bits do exist for patterns longer than  $\Delta = \log^{2+\epsilon} n$  (see [13]). Therefore, our basic strategy will be to have a separate data structure to solve queries of length  $p = \pi$ , for each  $\pi \in \{1, \dots, \Delta\}$ . Patterns with length  $p > \Delta$  can be handled with a single succinct data structure. More precisely, we design two different data structures that operate on top of a CSA:

- An  $O(n \log \log n / (\pi \log^\gamma n))$ -bits structure for handling queries of fixed length  $p = \pi$ , in time  $O(t_s(p) + k(t_{\text{SA}} + \log \log n + \log k) \pi \log^\gamma n)$ .
- An  $O(n/\log^\epsilon n + (n/\Delta) \log^2 n)$ -bits structure for handling queries with  $p > \Delta$  in time  $O(t_s(p) + \Delta(\Delta + t_{\text{SA}}) + k \log k \log^{2\epsilon} n (t_{\text{SA}} + \Delta \log^{1+\epsilon} n))$ .

By building the first structure for every  $\pi \in \{1, \dots, \Delta\}$ , any query can be handled using the appropriate structure. The  $\Delta$  structures for fixed pattern length add up to  $O(n(\log \log n)^2 / \log^\gamma n) = o(n/\log^{\gamma/2} n)$  bits, whereas that for long patterns uses  $O(n/\log^\epsilon n)$  bits. By choosing  $\epsilon = 4\epsilon = 2\gamma$ , the space is  $O(n/\log^{\epsilon/4} n)$  bits. As for the time, the structures for fixed  $p = \pi$  are most costly for  $\pi = \Delta$ , where their time is  $k(t_{\text{SA}} + \log \log n + \log k) \Delta \log^\gamma n$ . Adding up the time of the second structure, we get  $O(t_s(p) + \Delta(\Delta + k(t_{\text{SA}} + \log k \log^{1+\epsilon} n) \log^{2\epsilon} n))$ , which is upper bounded by  $O(t_s(p) + (\log^2 n + k(t_{\text{SA}} + \log k \log n)) \log^{2+\epsilon} n)$ . This yields Theorem 1.

Now we introduce some formalization to convey the key intuition. The term proximity  $\text{tp}(P, d)$  can be determined by just two occurrences of  $P$  in  $\mathbb{T}_d$ , which are the closest up to ties. We call them *critical occurrences*, and a pair of two closest occurrences is a *critical pair*. There can be multiple critical pairs.

**Definition 1.** *An integer  $i \in [1, n]$  is an occurrence of  $P$  in  $\mathbb{T}_d$  if the suffix  $\mathbb{T}[i..n]$  belongs to  $\mathbb{T}_d$  and  $\mathbb{T}[i..i+p-1] = P[1..p]$ . The set of all occurrences of  $P$  in  $\mathbb{T}$  is called  $\text{Occ}(P)$ .*

**Definition 2.** *An occurrence  $i_d$  of  $P$  in  $\mathbb{T}_d$  is a critical occurrence if there exists another occurrence  $i'_d$  of  $P$  in  $\mathbb{T}_d$  such that  $|i_d - i'_d| = \text{tp}(P, d)$ . The pair  $(i_d, i'_d)$  is called a critical pair of  $\mathbb{T}_d$  with respect to  $P$ .*

A key concept in our solution is that of *candidate sets* of occurrences, which contain sufficient information to solve the top- $k$  query (note that, due to ties, a top- $k$  query may have multiple valid answers).

**Definition 3.** Let  $\text{Topk}(P, k)$  be a valid answer for the top- $k$  query  $(P, k)$ . A set  $\text{Cand}(P, k) \subseteq \text{Occ}(P)$  is a candidate set of  $\text{Topk}(P, k)$  if, for each document identifier  $d \in \text{Topk}(P, k)$ , there exists a critical pair  $(i_d, i'_d)$  of  $\mathbb{T}_d$  with respect to  $P$  such that  $i_d, i'_d \in \text{Cand}(P, k)$ .

**Lemma 2.** Given a CSA on  $\mathcal{D}$ , a valid answer to query  $(P, k)$  can be computed from  $\text{Cand}(P, k)$  in  $O(z \log z)$  time, where  $z = |\text{Cand}(P, k)|$ .

*Proof.* Sort the set  $\text{Cand}(P, k)$  and traverse it sequentially. From the occurrences within each document  $\mathbb{T}_d$ , retain the closest consecutive pair  $(i_d, i'_d)$ , and finally report  $k$  documents with minimum values  $|i_d - i'_d|$ . This takes  $O(z \log z)$  time.

We show that this returns a valid answer set. Since  $\text{Cand}(P, k)$  is a candidate set, it contains a critical pair  $(i_d, i'_d)$  for  $d \in \text{Topk}(P, k)$ , so this critical pair (or another with the same  $|i_d - i'_d|$  value) is chosen for each  $d \in \text{Topk}(P, k)$ . If the algorithm returns an answer other than  $\text{Topk}(P, k)$ , it is because some document  $d \in \text{Topk}(P, k)$  is replaced by another  $d' \notin \text{Topk}(P, k)$  with the same score  $\text{tp}(P, d') = |i_{d'} - i'_{d'}| = |i_d - i'_d| = \text{tp}(d)$ .  $\square$

Our data structures aim to return a small candidate set (as close to size  $k$  as possible), from which a valid answer is efficiently computed using Lemma 2.

## 4 Data Structure for Queries with Fixed $p = \pi \leq \Delta$

We build an  $o(n/\pi)$ -bits structure for handling queries with pattern length  $p = \pi$ .

**Lemma 3.** There is an  $O(n \log \log n / (\pi \log^\gamma n))$ -bits data structure solving queries  $(P[1..p], k)$  with  $p = \pi$  in  $O(t_s(p) + k(t_{\text{SA}} + \log \log n + \log k) \pi \log^\gamma n)$  time.

The idea is to build an array  $F[1..n]$  such that a candidate set of size  $O(k)$ , for any query  $(P, k)$  with  $p = \pi$ , is given by  $\{\text{SA}[i], i \in [sp, ep] \wedge F[i] \leq k\}$ ,  $[sp, ep]$  being the suffix range of  $P$ . The key property to achieve this is that the ranges  $[sp, ep]$  are disjoint for all the patterns of a fixed length  $\pi$ . We build  $F$  as follows.

1. Initialize  $F[1..n] = n + 1$ .
2. For each pattern  $Q$  of length  $\pi$ ,
  - (a) Find the suffix range  $[\alpha, \beta]$  of  $Q$ .
  - (b) Find the list  $\mathbb{T}_{r_1}, \mathbb{T}_{r_2}, \mathbb{T}_{r_3}, \dots$  of documents in the ascending order of  $\text{tp}(Q, \cdot)$  values (ties broken arbitrarily).
  - (c) For each document  $\mathbb{T}_{r_\kappa}$  containing  $Q$  at least twice, choose a *unique* critical pair with respect to  $Q$ , that is, choose two elements  $j, j' \in [\alpha, \beta]$ , such that  $(i_{r_\kappa}, i'_{r_\kappa}) = (\text{SA}[j], \text{SA}[j'])$  is a critical pair of  $\mathbb{T}_{r_\kappa}$  with respect to  $Q$ . Then assign  $F[j] = F[j'] = \kappa$ .

The following observation is immediate.

**Lemma 4.** *For a query  $(P[1..p], k)$  with  $p = \pi$  and suffix array range  $[sp, ep]$  for  $P$ , the set  $\{SA[j], j \in [sp, ep] \wedge F[j] \leq k\}$  is a candidate set of size at most  $2k$ .*

*Proof.* A valid answer for  $(P, k)$  are the document identifiers  $r_1, \dots, r_k$  considered at construction time for  $Q = P$ . For each such document  $T_{r_\kappa}$ ,  $1 \leq \kappa \leq k$ , we have found a critical pair  $(i_{r_\kappa}, i'_{r_\kappa}) = (SA[j], SA[j'])$ , for  $j, j' \in [sp, ep]$ , and set  $F[j] = F[j'] = \kappa \leq k$ . All the other values of  $F[sp, ep]$  are larger than  $k$  (or  $\infty$ ). The size of the candidate set is thus at most  $2k$  (or less, if there are less than  $k$  documents where  $P$  occurs twice).  $\square$

However, we cannot afford to maintain  $F$  explicitly within the desired space bounds. Therefore, we replace  $F$  by a *sampled* array  $F'$ . The sampled array is built by cutting  $F$  into blocks of size  $\pi' = \pi \log^\gamma n$  and storing the logarithm of the minimum value for each block. This will increase the size of the candidate sets by a factor  $\pi'$ . More precisely,  $F'[1, n/\pi']$  is defined as

$$F'[j] = \lceil \log \min F[(j-1)\pi' + 1..j\pi'] \rceil.$$

Since  $F'[j] \in [0.. \log n]$ , the array can be represented using  $n \log \log n / \log^\gamma n$  bits. We maintain  $F'$  with a multiary wavelet tree [9], which maintains the space in  $O(n \log \log n / \log^\gamma n)$  bits and, since the alphabet size is logarithmic, supports in constant time operations *rank* and *select* on  $F'$ . Operation *rank* $(j, \kappa)$  counts the number of occurrences of  $\kappa$  in  $F'[1..j]$ , whereas *select* $(j, \kappa)$  gives the position of the  $j$ th occurrence of  $\kappa$  in  $F'$ .

**Query Algorithm.** To answer a query  $(P[1..p], k)$  with  $p = \pi$  using a CSA and  $F'$ , we compute the suffix range  $[sp, ep]$  of  $P$  in time  $t_s(p)$ , and then do as follows.

1. Among all the blocks of  $F$  overlapping the range  $[sp, ep]$ , identify those containing an element  $\leq 2^{\lceil \log k \rceil}$ , that is, compute the set

$$S_{blocks} = \{j, \lceil sp/\pi' \rceil \leq j \leq \lceil ep/\pi' \rceil \wedge F'[j] \leq \lceil \log k \rceil\}.$$

2. Generate  $\text{Cand}(P, k) = \{SA[j'], j \in S_{blocks} \wedge j' \in [(j-1)\pi' + 1, j\pi']\}$ .
3. Find the query output from the candidate set  $\text{Cand}(P, k)$ , using Lemma 2.

For step 1, the wavelet tree representation of  $F'$  generates  $S_{blocks}$  in time  $O(1 + |S_{blocks}|)$ : All the  $2^t$  positions<sup>3</sup>  $j \in [sp, ep]$  with  $F'[j] = t$  are  $j = \text{select}(\text{rank}(sp-1, t) + i, t)$  for  $i \in [1, 2^t]$ . We notice if there are no sufficient documents if we obtain a  $j > ep$ , in which case we stop.

The set  $\text{Cand}(P, k)$  is a candidate set of  $(P, k)$ , since any  $j \in [sp, ep]$  with  $F[j] \leq k$  belongs to some block of  $S_{blocks}$ . Also the number of  $j \in [sp, ep]$  with  $F[j] \leq 2^{\lceil \log k \rceil}$  is at most  $2 \cdot 2^{\lceil \log k \rceil} \leq 4k$ , therefore  $|S_{blocks}| \leq 4k$ .

Now,  $\text{Cand}(P, k)$  is of size  $|S_{blocks}| \pi' = O(k\pi')$ , and it is generated in step 2 in time  $O(k t_{SA} \pi')$ . Finally, the time for generating the final output using Lemma 2 is  $O(k\pi' \log(k\pi')) = O(k\pi \log^\gamma n (\log k + \log \log n + \log \pi))$ . By considering that  $\pi \leq \Delta = \log^{2+\epsilon} n$ , we obtain Lemma 3.

<sup>3</sup> Except for  $t = 0$ , which has 2 positions.

## 5 Data Structure for Queries with $p > \Delta$

We prove the following result in this section.

**Lemma 5.** *There is an  $O(n/\log^\epsilon n + (n/\Delta) \log^2 n)$ -bits structure solving queries  $(P[1..p], k)$ , with  $p > \Delta$ , in  $O(t_s(p) + \Delta(\Delta + t_{SA}) + k \log k \log^{2\epsilon} n(t_{SA} + \Delta \log^{1+\epsilon} n))$  time.*

We start with a concept similar to that of a candidate set, but weaker in the sense that it is required to contain only one element of each critical pair.

**Definition 4.** *Let  $\text{Topk}(P, k)$  be a valid answer for the top- $k$  query  $(P, k)$ . A set  $\text{Semi}(P, k) \subseteq [n]$  is a semi-candidate set of  $\text{Topk}(P, k)$  if it contains at least one critical occurrence  $i_d$  of  $P$  in  $\mathsf{T}_d$  for each document identifier  $d \in \text{Topk}(P, k)$ .*

Our structure in this section generates a semi-candidate set  $\text{Semi}(P, k)$ . Then, a candidate set  $\text{Cand}(P, k)$  is generated as the union of  $\text{Semi}(P, k)$  and the set of occurrences of  $P$  that are immediately before and immediately after every position  $i \in \text{Semi}(P, k)$ . This is obviously a valid candidate set. Finally, we apply Lemma 2 on  $\text{Cand}(P, k)$  to compute the final output.

### 5.1 Generating a Semi-candidate Set

This section proves the following result.

**Lemma 6.** *A structure of  $O(n(\log \log n)^2 / \log^\delta n)$  bits plus a CSA can generate a semi-candidate set of size  $O(k \log k \log^\delta n)$  in time  $O(t_{SA} k \log k \log^\delta n)$ .*

Let  $\text{Leaf}(x)$  (resp.,  $\text{Leaf}(y)$ ) be the set of leaves in the subtree of node  $x$  (resp.,  $y$ ) in GST,  $\text{Leaf}(x \setminus y) = \text{Leaf}(x) \setminus \text{Leaf}(y)$ . The following lemma holds.

**Lemma 7.** *The set  $\text{Semi}(\text{path}(y), k) \cup \{\text{SA}[j], \ell_j \in \text{Leaf}(x \setminus y)\}$  is a semi-candidate set of  $(\text{path}(x), k)$ .*

*Proof.* Let  $d \in \text{Topk}(\text{path}(x), k)$ , then our semi-candidate set should contain  $i_d$  or  $i'_d$  for some critical pair  $(i_d, i'_d)$ . If there is some such critical pair where  $i_d$  or  $i'_d$  are occurrences of  $\text{path}(x)$  but not of  $\text{path}(y)$ , then  $\ell_j$  or  $\ell_{j'}$  are in  $L(x \setminus y)$ , for  $\text{SA}[j] = i_d$  and  $\text{SA}[j'] = i'_d$ , and thus our set contains it. If, on the other hand, both  $i_d$  and  $i'_d$  are occurrences of  $\text{path}(y)$  for all critical pairs  $(i_d, i'_d)$ , then  $\text{tp}(\text{path}(y), d) = \text{tp}(\text{path}(x), d)$ , and the critical pairs of  $\text{path}(x)$  are the critical pairs of  $\text{path}(y)$ . Thus  $\text{Semi}(y, k)$  contains  $i_d$  or  $i'_d$  for some such critical pair.  $\square$

Our approach is to precompute and store  $\text{Semi}(\text{path}(y), k)$  for carefully selected nodes  $y \in \text{GST}$  and  $k$  values, so that any arbitrary  $\text{Semi}(\text{path}(x), k)$  set can be computed efficiently. The succinct framework of Hon et al. [16] is adequate for this.

*Node Marking Scheme.* The idea [16] is to mark a set  $\text{Mark}_g$  of nodes in GST based on a *grouping factor*  $g$ : Every  $g$ th leaf is marked, and the LCA of any two consecutive marked leaves is also marked. Then the following properties hold.

1.  $|\text{Mark}_g| \leq 2n/g$ .
2. If there exists no marked node in the subtree of  $x$ , then  $|\text{Leaf}(x)| < 2g$ .
3. If it exists, then the highest marked descendant node  $y$  of any unmarked node  $x$  is unique, and  $|\text{Leaf}(x \setminus y)| < 2g$ .

We use this idea, and a later refinement [14]. Let us first consider a variant of Lemma 6 where  $k = \kappa$  is fixed at construction time. We use a CSA and an  $O(n/\log^\delta n)$ -bit CST on it, see Section 2. We choose  $g = \kappa \log \kappa \log^{1+\delta} n$  and, for each node  $y \in \text{Mark}_g$ , we explicitly store a candidate set  $\text{Semi}(\text{path}(y), \kappa)$  of size  $\kappa$ . The space required is  $O(|\text{Mark}_g| \kappa \log n) = O(n/(\log \kappa \log^\delta n))$  bits.

To solve a query  $(P, \kappa)$ , we find the suffix range  $[sp, ep]$ , then the locus node of  $P$  is  $x = \text{LCA}(\ell_{sp}, \ell_{ep})$ . Then we find  $y = \text{LCA}(\ell_{g \lceil sp/g \rceil}, \ell_{g \lfloor ep/g \rfloor})$ , the highest marked node in the subtree of  $x$ . Then, by the given properties of the marking scheme, combined with Lemma 7, a semi-candidate set of size  $O(g + \kappa) = O(\kappa \log \kappa \log^{1+\delta} n)$  can be generated in  $O(t_{\text{SA}} \kappa \log \kappa \log^{1+\delta} n)$  time.

To reduce this time, we employ dual marking scheme [14]. We identify a larger set  $\text{Mark}_{g'}$  of nodes, for  $g' = \kappa \log \kappa \log^\delta n$ . To avoid confusion, we call these *prime* nodes, not marked nodes. For each node  $y' \in \text{Mark}_{g'}$ , we precompute a candidate set  $\text{Semi}(\text{path}(y'), \kappa)$  of size  $\kappa$ . Let  $y$  be the (unique) highest marked node in the subtree of  $y'$ . Then we store  $\kappa$  bits in  $y'$  to indicate which of the  $\kappa$  nodes stored in  $\text{Semi}(\text{path}(y), \kappa)$  also belong to  $\text{Semi}(\text{path}(y'), \kappa)$ . By the same proof of Lemma 7, elements in  $\text{Semi}(\text{path}(y'), \kappa) \setminus \text{Semi}(\text{path}(y), \kappa)$  must have a critical occurrence in  $\text{Leaf}(y' \setminus y)$ . Then, instead of explicitly storing the critical positions  $i_d \in \text{Semi}(\text{path}(y'), \kappa) \setminus \text{Semi}(\text{path}(y), \kappa)$ , we store their left-to-right position in  $\text{Leaf}(y' \setminus y)$ . Storing  $\kappa$  such positions in leaf order requires  $O(\kappa \log(g/\kappa)) = O(\kappa \log \log n)$  bits, using for example gamma codes. The total space is  $O(|\text{Mark}_{g'}| \kappa \log \log n) = O(n \log \log n / (\log \kappa \log^\delta n))$  bits.

Now we can apply the same technique to obtain a semi-candidate set from  $\text{Mark}_{g'}$ , yet of smaller size  $O(g' + \kappa) = O(\kappa \log \kappa \log^\delta n)$ , in time  $O(t_{\text{SA}} \kappa \log \kappa \log^\delta n)$ .

We are now ready to complete the proof Lemma 6. We maintain structures as described for all the values of  $\kappa$  that are powers of 2, in total  $O((n \log \log n / \log^\delta n) \cdot \sum_{i=1}^{\log D} 1/i) = O(n(\log \log n)^2 / \log^\delta n)$  bits of space. To solve a query  $(P, k)$ , we compute  $\kappa = 2^{\lceil \log k \rceil} < 2k$  and return the semi-candidate set of  $(P, \kappa)$  using the corresponding structure.

## 5.2 Generating the Candidate Set

The problem boils down to the task of, given  $P[1..p]$  and an occurrence  $q$ , finding the occurrence of  $P$  closest to  $q$ . In other words, finding the first and the last occurrence of  $P$  in  $\text{T}[q + 1..n]$  and  $\text{T}[1..q + p - 1]$ , respectively. We employ suffix sampling to obtain the desired space-efficient structure. The idea is to exploit the fact that, if  $p > \Delta$ , then for every occurrence  $q$  of  $P$  there must be an integer  $j = \Delta \lceil q/\Delta \rceil$  (a multiple of  $\Delta$ ) and  $t \leq \Delta$ , such that  $P[1..t]$  is a suffix of  $\text{T}[1..j]$  and  $P[t + 1..p]$  is a prefix of  $\text{T}[j + 1..n]$ . We call  $q$  an *offset- $t$  occurrence* of  $P$ . Then,  $\text{Cand}(P, k)$  can be computed as follows:

1. Find  $\text{Semi}(P, k)$  using Lemma 6.
2. For each  $q \in \text{Semi}(P, k)$  and  $t \in [1, \Delta]$ , find the offset- $t$  occurrences of  $P$  that are immediately before and immediately after  $q$ .
3. The occurrences found in the previous step, along with the elements in  $\text{Semi}(P, k)$ , constitute  $\text{Cand}(P, k)$ .

In order to perform step 2 efficiently, we maintain the following structures.

- **Sparse Suffix Tree (SST)**: A suffix  $\text{T}[\Delta i + 1..n]$  is a *sparse suffix*, and the trie of all sparse suffixes is a *sparse suffix tree*. The *sparse suffix range* of a pattern  $Q$  is the range of the sparse suffixes in SST that are prefixed by  $Q$ . Given the suffix range  $[sp, ep]$  of a pattern, its sparse suffix range  $[ssp, sep]$  can be computed in constant time by maintaining a bitmap  $B[1..n]$ , where  $B[j] = 1$  iff  $\text{T}[\text{SA}[j]..n]$  is a sparse suffix. Then  $ssp = 1 + \text{rank}_B(sp - 1)$  and  $sep = \text{rank}_B(sp)$ . Since  $B$  has  $n/\Delta$  1s, it can be represented in  $O((n/\Delta) \log \Delta)$  bits while supporting  $\text{rank}_B$  operation in constant time for any  $\Delta = O(\text{polylog } n)$  [28].
- **Sparse Prefix Tree (SPT)**: A prefix  $\text{T}[1..\Delta i]$  is a *sparse prefix*, and the trie of the *reverses* of all sparse prefixes is a *sparse prefix tree*. The *sparse prefix range* of a pattern  $Q$  is the range of the sparse prefixes in SPT with  $Q$  as a suffix. The SPT can be represented as a blind trie [7] using  $O((n/\Delta) \log n)$  bits. Then the search for the sparse prefix range of  $Q$  can be done in  $O(|Q|)$  time, by descending using the reverse of  $Q^4$ . Note that the blind trie may return a fake node when  $Q$  does not exist in the SPT.
- **Orthogonal Range Successor/Predecessor Search Structure** over a set of  $\lceil n/\Delta \rceil$  points of the form  $(x, y, z)$ , where the  $y$ th leaf in SST corresponds to  $\text{T}[x..n]$  and the  $z$ th leaf in SPT corresponds to  $\text{T}[1..(x - 1)]$ . The space needed is  $O((n/\Delta) \log^2 n)$  bits (recall Lemma 1).

The total space of the structures is  $O((n/\Delta) \log^2 n)$  bits. They allow computing first offset- $t$  occurrence of  $P$  in  $\text{T}[q + 1..n]$  as follows: find  $[ssp_t, sep_t]$  and  $[ssp'_t, sep'_t]$ , the sparse suffix range of  $P[t + 1..p]$  and the sparse prefix range of  $P[1..t]$ , respectively. Then, using an orthogonal range successor query, find the point  $(e, \cdot, \cdot)$  with the lowest  $x$ -coordinate value in  $[q + t + 1, n] \times [ssp_t, sep_t] \times [ssp'_t, sep'_t]$ . Then,  $e - t$  is the answer. Similarly, the last offset- $t$  occurrence of  $P$  in  $\text{T}[1..q - 1]$  is  $f - t$ , where  $(f, \cdot, \cdot)$  is the point in  $[1, q + t - 1] \times [ssp_t, sep_t] \times [ssp'_t, sep'_t]$  with the highest  $x$ -coordinate value.

First, we compute all the ranges  $[ssp_t, sep_t]$  using the SST. This requires knowing the interval  $\text{SA}[sp_t, ep_t]$  of  $P[t + 1..p]$  for all  $1 \leq t \leq \Delta$ . We compute these by using the CSA to search for  $P[\Delta + 1..p]$  (in time at most  $t_s(p)$ ), which gives  $[sp_\Delta, ep_\Delta]$ , and then computing  $[sp_{t-1}, ep_{t-1}] = \text{Wlink}(P[t], [sp_t, ep_t])$  for  $t = \Delta - 1, \dots, 1$ . Using an  $o(n)$ -bits CST (see Section 2), this takes  $O(\Delta t_{\text{SA}})$  time. Then the SST finds all the  $[ssp_t, sep_t]$  values in time  $O(\Delta)$ . Thus the time spent on the SST searches is  $O(t_s(p) + \Delta t_{\text{SA}})$ .

<sup>4</sup> Using perfect hashing to move in constant time towards the children.

Second, we search the SPT for reverse pattern prefixes of lengths 1 to  $\Delta$ , and thus they can all be searched for in time  $O(\Delta^2)$ . Since the SPT is a blind trie, it might be either that the intervals  $[ssp'_t, sep'_t]$  it returns are the correct interval of  $P[1..t]$ , or that  $P[1..t]$  does not terminate any sparse prefix. A simple way to determine which is the case is to perform the orthogonal range search as explained, asking for the successor  $e_0$  of position 1, and check whether the resulting position,  $e_0 - t$ , is an occurrence of  $P$ , that is, whether  $SA^{-1}[e_0 - t] \in [sp, ep]$ . This takes  $O(t_{SA} + \log^{1+\epsilon} n)$  time per verification. Considering the searches plus verifications, the time spent on the SPT searches is  $O(\Delta(\Delta + t_{SA} + \log^{1+\epsilon} n))$ .

Finally, after determining all the intervals  $[ssp_t, sep_t]$  and  $[ssp'_t, sep'_t]$ , we perform  $O(|\text{Semi}(P, k)|\Delta)$  orthogonal range searches for positions  $q$ , in time  $O(|\text{Semi}(P, k)|\Delta \log^{1+\epsilon} n)$ , and keep the closest one for each  $q$ .

**Lemma 8.** *Given a semi-candidate set  $\text{Semi}(P, k)$ , where  $p > \Delta$ , a candidate set  $\text{Cand}(P, k)$  of size  $O(|\text{Semi}(P, k)|)$  can be computed in time  $O(t_s(p) + \Delta(\Delta + t_{SA} + |\text{Semi}(P, k)| \log^{1+\epsilon} n))$  using a data structure of  $O((n/\Delta) \log^2 n)$  bits.*

Thus, by combining Lemma 6 using  $\delta = 2\epsilon$  (so its space is  $o(n/\log^\epsilon n)$  bits) and Lemma 8, we obtain Lemma 5.

## 6 Concluding Remarks

We have obtained the first succinct result for top- $k$  term-proximity queries. The following additional results will be presented in the full version of this paper.

1. Another trade-off for top- $k$  term-proximity queries with space and query time  $2n \log \sigma + o(n \log \sigma) + O(n \log \log n)$  bits and  $O(p + k \log k \log^{1+\epsilon} n)$ , respectively. Notice that, when  $\log \log n = o(\log \sigma)$ , the trade-off matches with the best known result for top- $k$  term-frequency queries [15].
2. In a more realistic scenario,  $\text{score}(\cdot, \cdot)$  is a weighted sum of PageRank, term-frequency and term-proximity with predefined non-negative weights [33]. Top- $k$  queries with such ranking functions can be handled using an index of space  $2n \log \sigma + o(n \log \sigma)$  bits in time  $O(p + k \log k \log^{4+\epsilon} n)$ .

## References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval, 2nd edn. Addison-Wesley (2011)
2. Belazzougui, D., Navarro, G.: Alphabet-independent compressed text indexing. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 748–759. Springer, Heidelberg (2011)
3. Belazzougui, D., Navarro, G., Valenzuela, D.: Improved compressed indexes for full-text document retrieval. J. Discr. Alg. **18**, 3–13 (2013)
4. Benson, G., Waterman, M.: A fast method for fast database search for all  $k$ -nucleotide repeats. Nucleic Acids Research **22**(22) (1994)



5. Broschart, A., Schenkel, R.: Index tuning for efficient proximity-enhanced query processing. In: Geva, S., Kamps, J., Trotman, A. (eds.) INEX 2009. LNCS, vol. 6203, pp. 213–217. Springer, Heidelberg (2010)
6. Büttcher, S., Clarke, C.L.A., Cormack, G.: Information Retrieval: Implementing and Evaluating Search Engines. MIT Press (2010)
7. Ferragina, P., Grossi, R.: The string B-tree: A new data structure for string search in external memory and its applications. *J. ACM* **46**(2), 236–280 (1999)
8. Ferragina, P., Manzini, G.: Indexing compressed text. *J. ACM* **52**(4), 552–581 (2005)
9. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. *ACM Trans. Alg.* **3**(2), art. 20 (2007)
10. Gagie, T., Navarro, G., Puglisi, S.J.: New algorithms on wavelet trees and applications to information retrieval. *Theor. Comp. Sci.* **426–427**, 25–41 (2012)
11. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.* **35**(2), 378–407 (2005)
12. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997)
13. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: On position restricted substring searching in succinct space. *J. Discr. Alg.* **17**, 109–114 (2012)
14. Hon, W.-K., Shah, R., Thankachan, S.V., Vitter, J.S.: Faster compressed top- $k$  document retrieval. In: Proc. 23rd DCC, pp. 341–350 (2013)
15. Hon, W.-K., Shah, R., Thankachan, S.V., Scott Vitter, J.: Space-efficient frameworks for top- $k$  string retrieval. *J. ACM* **61**(2), 9 (2014)
16. Hon, W.-K., Shah, R., Vitter, J.S.: Space-efficient framework for top- $k$  string retrieval problems. In: Proc. 50th FOCS, pp. 713–722 (2009)
17. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. *SIAM J. Comp.* **22**(5), 935–948 (1993)
18. Munro, J.I., Navarro, G., Shah, R., Thankachan, S.V.: Ranked document selection. In: Ravi, R., Gørtz, I.L. (eds.) SWAT 2014. LNCS, vol. 8503, pp. 344–356. Springer, Heidelberg (2014)
19. Muthukrishnan, S.: Efficient algorithms for document retrieval problems. In: Proc. 13th SODA, pp. 657–666 (2002)
20. Navarro, G.: Spaces, trees and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comp. Surv.* **46**(4), art. 52 (2014)
21. Navarro, G., Mäkinen, V.: Compressed full-text indexes. *ACM Comp. Surv.* **39**(1), art. 2 (2007)
22. Navarro, G., Nekrich, Y.: Top- $k$  document retrieval in optimal time and linear space. In: Proc. 23rd SODA, pp. 1066–1078 (2012)
23. Navarro, G., Russo, L.: Fast fully-compressed suffix trees. In: Proc. 24th DCC, pp. 283–291 (2014)
24. Navarro, G., Thankachan, S.V.: Faster top- $k$  document retrieval in optimal space. In: Kurland, O., Lewenstein, M., Porat, E. (eds.) SPIRE 2013. LNCS, vol. 8214, pp. 255–262. Springer, Heidelberg (2013)
25. Navarro, G., Thankachan, S.V.: Top- $k$  document retrieval in compact space and near-optimal time. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) Algorithms and Computation. LNCS, vol. 8283, pp. 394–404. Springer, Heidelberg (2013)
26. Navarro, G., Thankachan, S.V.: New space/time tradeoffs for top- $k$  document retrieval on sequences. *Theor. Comput. Sci.* **542**, 83–97 (2014)
27. Nekrich, Y., Navarro, G.: Sorted range reporting. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 271–282. Springer, Heidelberg (2012)

28. Pătraşcu, M.: Succincter. In: Proc. 49th FOCS, pp. 305–313 (2008)
29. Schenkel, R., Broschart, A., Hwang, S., Theobald, M., Weikum, G.: Efficient text proximity search. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 287–299. Springer, Heidelberg (2007)
30. Shah, R., Sheng, C., Thankachan, S.V., Vitter, J.S.: Top- $k$  document retrieval in external memory. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 803–814. Springer, Heidelberg (2013)
31. Weiner, P.: Linear pattern matching algorithm. In: Proc. 14th Annual IEEE Symposium on Switching and Automata Theory, pp. 1–11 (1973)
32. Yan, H., Shi, S., Zhang, F., Suel, T., Wen, J.-R.: Efficient term proximity search with term-pair indexes. In: CIKM, pp. 1229–1238 (2010)
33. Zhu, M., Shi, S., Li, M., Wen, J.-R.: Effective top- $k$  computation in retrieving structured documents with term-proximity support. In: CIKM, pp. 771–780 (2007)
34. Zhu, M., Shi, S., Yu, N., Wen, J.-R.: Can phrase indexing help to process non-phrase queries? In: CIKM, pp. 679–688 (2008)

# The Power and Limitations of Static Binary Search Trees with Lazy Finger

Presenjit Bose<sup>1</sup>, Karim Douïeb<sup>3</sup>, John Iacono<sup>2(✉)</sup>, and Stefan Langerman<sup>3</sup>

<sup>1</sup> Carleton University, Ottawa, ON, Canada

<sup>2</sup> New York University, Shanghai, China

iacono@nyu.edu

<sup>3</sup> Université Libre de Bruxelles, Brussels, Belgium

**Abstract.** A static binary search tree where every search starts from where the previous one ends (*lazy finger*) is considered. Such a search method is more powerful than that of the classic optimal static trees, where every search starts from the root (*root finger*), and less powerful than when rotations are allowed—where finding the best rotation based tree is the topic of the dynamic optimality conjecture of Sleator and Tarjan. The runtime of the classic root-finger tree can be expressed in terms of the entropy of the distribution of the searches, but we show that this is not the case for the optimal lazy finger tree. A non-entropy based asymptotically-tight expression for the runtime of the optimal lazy finger trees is derived, and a dynamic programming-based method is presented to compute the optimal tree.

## 1 Introduction

**Static Trees.** A binary search tree is one of the most fundamental data structures in computer science. In response to a search operation, some binary trees perform changes in the data structure, while others do not. For example, the splay tree [18] data structure performs a sequence of rotations that moves the searched item to the root. Other binary search tree data structures do not change at all during a search, for example, red-black trees [13] and AVL trees [1]. We will call BSTs that do not perform changes in the structure during searches to be *static* and call trees that perform changes *BSTs with rotations*. In this work we do not consider insertions and deletions, only searches in the comparison model, and thus can assume without loss of generality that all structures under consideration are storing the integers from 1 to  $n$  and that all searches are to these items.

---

Research for P. Bose supported in part by NSERC.

Research partially completed at NYU School of Engineering with support from NSF grant CCF-1319648. Research partially completed at Université Libre de Bruxelles with support from FNRS. Research partially completed at and supported by MADALGO, a center of the Danish National Research Foundation.

S. Langerman is Directeur de Recherches du F.R.S.-FNRS.

We consider two variants of static BSTs: root finger and lazy finger. In the classic method, the *root finger* method, the first search proceeds from the root to the item being searched. In the second and subsequent searches, a root finger BST executes the searches in the same manner, always starting each search from the root. In contrast, here we consider *lazy finger* BSTs to be those which start each search at the destination of the previous search and move to the item being searched. In general, this movement involves going up to the least common ancestor (LCA) of the previous and current items being searched, and then moving down from the LCA to the current item being searched. In order to facilitate such a search, each node of the tree needs to be augmented with the minimal and maximal elements in its subtree.

**Notation and Definitions.** A static tree  $T$  is a fixed binary search tree containing  $n$  elements. No rotations are allowed. The data structure must process a sequence of searches, by moving a single pointer in the tree. Let  $r(T, i, j)$  be the time to move the pointer in the tree  $T$  from node  $i$  to  $j$ . If  $d_T(i)$  represents the depth of node  $i$ , with the root defined as having depth zero, then

$$\begin{aligned} r(T, i, j) &= d_T(i) - d_T(\text{LCA}_T(i, j)) + d_T(j) - d_T(\text{LCA}_T(i, j)) \\ &= d_T(i) + d_T(j) - 2d_T(\text{LCA}_T(i, j)). \end{aligned}$$

The runtime to execute a sequence  $X = x_1, x_2, \dots, x_m$  of searches on a tree  $T$  using the root finger method is

$$R_{\text{root}}(T, X) = \sum_{i=1}^m r(T, \text{root}(T), x_i) = \sum_{i=1}^m d_T(x_i)$$

and the runtime to execute the same sequence on a tree  $T$  using the lazy finger method is

$$R_{\text{lazy}}(T, X) = \sum_{i=1}^m r(T, x_{i-1}, x_i) = \left( 2 \sum_{i=1}^m (d_T(x_i) - d_T(\text{LCA}_T(x_i, x_{i-1}))) \right) - d_T(x_m)$$

where  $x_0$  is defined to be the root of  $T$ , which is where the first search starts.

**History of Optimal Static Trees with Root Finger.** For the root finger method, once the tree  $T$  is fixed, the cost of any single search in tree  $T$  depends only on the search and the tree, not on any of the search history. Thus, the optimal search tree for the root finger method is a function only of the frequency of the searches for each item. Let  $f_X(a)$  denote the number of searches in  $X$  to  $a$ . Given  $f_X$ , computing the optimal static BST with root finger has a long history. In 1971, Knuth gave a  $O(n^2)$  dynamic programming solution that finds the optimum tree [15]. More interestingly is the discovery of a connection between the runtime of the optimal tree and the entropy of the frequencies:

$$H(f_X) = \sum_{a=1}^n \frac{f_X(a)}{m} \lg \frac{m}{f_X(a)}.$$

Melhorn [16] showed that a simple greedy heuristic proposed by Knuth [15] and shown to have a linear-time implementation by Fredman [11] produced a static tree where an average search takes time  $2 + \frac{1}{1 - \lg(\sqrt{5} - 1)} H(f_X)$ . Furthermore, Melhorn demonstrated a lower bound of  $\frac{1}{\lg 3} H(f_X)$  for an average search in an optimal static tree, and showed this bound was tight for infinitely many distributions. Thus, by 1975, it was established that the runtime for an average search in an optimal search tree with root finger was  $\Theta(H(f_X))$ , and that such a tree could easily be computed in linear time.

**Our Results.** We wish to study the natural problem of what we have coined search with a lazy finger in a static tree, i.e. have each search start where the last one ended. We seek to characterize the optimal tree for this search strategy, and describe how to build it.

The lazy finger method is asymptotically clearly no worse than the root finger method; moving up to the LCA and back down is better than moving to the root and back down, which is exactly double the cost of the root finger method. But, in general, is the lazy finger method better? For the lazy finger method, the cost of a single search in a static tree depends only on the current search and the previous search—this puts lazy finger’s runtime dependence on the search sequence between that of root finger and trees with rotations. Thus the optimal search tree for the lazy finger method only depends on the frequency of each search transition; let  $f_X(a, b)$  be the number of searches in  $X$  to  $b$  where the previous search was to  $a$ . Given these pairwise frequencies (from which the frequencies  $f_X(a)$  can easily be computed), is there a nice closed form for the runtime of the optimal BST with lazy finger? One natural runtime to consider is the conditional entropy:

$$H_c(f_X) = \sum_{a=1}^n \sum_{b=1}^n \frac{f_X(a, b)}{m} \lg \frac{f_X(a)}{f_X(a, b)}$$

This is of interest as information theory gives this as an expected lower bound<sup>1</sup> if the search sequence is derived from a Markov chain where  $n$  states represents searching each item.

While a runtime related to the conditional entropy is the best achievable by any algorithm parameterized solely on the pairwise frequencies, however, we will show in Lemma 5 that the conditional entropy is impossible to be asymptotically achieved for any BST, static or dynamic, within any  $o(\log n)$  factor. Thus, for the root finger, the lower bound given by information theory is achievable, yet

---

<sup>1</sup> When multiplied by  $\frac{1}{\lg 3}$ , as the information theory lower bound holds for binary decisions and as observed in [16] needs to be adjusted to the ternary decisions that occur at each node when traversing a BST.

for lazy finger it is not related to the runtime of the optimal tree. In Section 7 we will present a simple static non-tree structure whose runtime is related to the conditional entropy.

This still leaves us with the question: is there a simple closed form for the runtime of the optimal BST with lazy finger? We answer this in the affirmative by showing an equivalence between the runtime of BSTs with lazy finger and something known as the weighted dynamic finger runtime. In the weighted dynamic finger runtime, if item  $i$  is given weight  $w_i$ , then the time to execute search  $x_i$  is  $\lg \frac{\sum_{k=\min(x_i, x_{i-1})}^{\max(x_i, x_{i-1})} w_k}{\min(w_{x_{i-1}}, w_{x_i})}$ . Our main theorem is that the runtime of the best static tree with lazy finger,  $LF(X)$ , is given by the weighted dynamic finger runtime bound with the best choice of weights:

$$LF(X) = \min_T R_{\text{lazy}}(T, X) = \Theta \left( \min_W \left\{ \sum_{i=1}^m \lg \frac{\sum_{k=\min(x_i, x_{i-1})}^{\max(x_i, x_{i-1})} w_k}{\min(w_{x_{i-1}}, w_{x_i})} \right\} \right)$$

To prove this, we first state the result of Seidel and Aragon [17] in Section 2 of how to construct a tree with the weighted dynamic finger runtime given a set of weights. Then, in Section 3, we show how, given any static tree  $T$ , there exists weights such that the runtime of  $T$  on a sequence using lazy finger can be lower bounded using the weighted dynamic finger runtime with these weights. These results are combined in Section 4 to give the main theorem.

While a nice closed-form formula for the runtime of splay trees is not known, there are several different bounds on their runtime: working set, static finger, dynamic finger, and static optimality [7, 8, 18]. One implication of our result is that the runtime of the optimal lazy finger tree is asymptotically as good as that of all of the aforementioned bounds with the exception of the working set bound (see Theorem 3 for why the working set bound does not hold on a lazy finger static structure).

While these results have served to characterize the best runtime for the optimal BST, a concrete method is needed to compute the best tree given the pairwise frequencies. We present a dynamic programming solution in Section 6; this solution takes time  $O(n^3)$  to compute the optimal tree for lazy finger, given a table of size  $n^2$  with the frequency of each pair of searches occurring adjacently. This method could be extended using the ideas of Iacono and Mulzer [14] into one which periodically rebuilds the static structure using the observed frequencies so far; the result would be an online structure that for sufficiently long search sequences achieves a runtime that is within a constant factor of the optimal tree without needing to be initialized with the pairwise frequencies.

**Relation to Finger Search Structures.** The results here have a relation to the various finger search structures that have been proposed. We note, first of all, that the trees we are considering are not level linked; the only pointers are to the parent and children. Secondly, while the basic finger search runtime of  $O(\sum_{i=2}^m \log |x_i - x_{i-1}|)$  (recall that we are assuming the  $x_i$  are integers from

1 to  $n$ ) is long known to be easily achievable in a static tree, it is easily shown that there are some search sequences  $X$  for which the optimal tree performs far better. For example, the search sequence  $x_i = i\sqrt{n} \bmod n$  where  $n$  is a perfect square can be easily executed in time  $O(m)$  on the best static tree with lazy finger, which is much better than the  $O(m \log n)$  of dynamic finger.

But this limitation of the  $O(\sum_{i=2}^m \log |x_i - x_{i-1}|)$  runtime has been long known, which is why the weighted version of finger search was proposed. Our main contribution is to realize that the weighted dynamic finger runtime bound, which was not proposed in the context of lazy finger, is the asymptotically tight characterization of BSTs with lazy finger when used with the best choice of weights.

**Why Static Trees?** Static trees are less powerful than dynamic ones in terms of the classes of search sequence distributions that can be executed quickly, so why are we studying them? One should use the simplest structure with the least overhead that gets the job done. By completely categorizing the runtime of the optimal tree with lazy finger, one can know if such a structure is appropriate for a particular application or whether one should instead use the more powerful dynamic trees, or simpler root-finger trees.

Rotation-based trees have horrible cache performance. However, there are methods to map the nodes of a static tree to memory so as to have optimal performance in the disk-access model and cache-oblivious models of the memory hierarchy [6, 9, 12, 19]. One leading cache oblivious predecessor query data structure that supports insertion and deletion works by having a static tree and moves the data around in the fixed static tree in response to insertions and deletions and only periodically rebuilds the static structure [4]—in such a structure an efficient static structure is the key to obtaining good performance even with insertions and deletions.

Also, concurrency becomes a real issue in dynamic trees, which requires another layer of complexity to resolve (see, for example Bronson et al. [5]), while static trees trivially support concurrent operations.

## 2 Weights Give a Tree

**Theorem 1 (Seidel and Aragon [3]).** *Given a set of positive weights  $W = w_1, w_2, \dots, w_n$ , there is a randomized method to choose a tree  $T_W$  such that the expected runtime is  $r(T_W, i, j) = O\left(\lg \frac{\sum_{k=\min(i,j)}^{\max(i,j)} w_k}{\min(w_i, w_j)}\right)$ .*

The method to randomly create  $T_W$  is a straightforward random tree construction using the weights: recursively pick the root using the normalized weights of all nodes as probabilities. Thus, by the probabilistic method [2], there is a deterministic tree, call it  $T_W$  whose runtime over the sequence  $X$  is at most the runtime bound of Seidel and Aragon for the sequence  $X$  on the best possible choice of weights.

**Corollary 1.** *For any set of positive weights  $W = w_1, w_2, \dots, w_n$  there is a tree  $T_W(X)$  such that*

$$\sum_{i=1}^m r(T_W(X), x_{i-1}, x_i) = O \left( \min_W \left\{ \sum_{i=1}^m \lg \frac{\sum_{k=\min(x_i, x_{i-1})}^{\max(x_i, x_{i-1})} w_k}{\min(w_{x_{i-1}}, w_{x_i})} \right\} \right)$$

*Proof.* This follows directly from Seidel and Aragon, where  $T_W(X)$  is a tree that achieves the expected runtime of their randomized method for the best choice of weights. □

### 3 Trees Can Be Represented by Weights

**Lemma 1.** *For each tree  $T$  there is a set of weights  $W^T = w_1^T, w_2^T, \dots, w_n^T$  such that for all  $i, j$   $r(T, i, j) = \Theta \left( \lg \frac{\sum_{k=\min(i,j)}^{\max(i,j)} w_k^T}{\min(w_i^T, w_j^T)} \right)$ .*

*Proof.* These weights are simple: give a node at depth  $d$  in  $T$  a weight of  $\frac{1}{4^d}$ . Consider a search that starts at node  $i$  and goes to node  $j$ . Such a path goes up from  $i$  to  $\text{LCA}_T(i, j)$  and down to  $j$ . A lower bound on  $\sum_{k=\min(i,j)}^{\max(i,j)} w_k^T$  is the weight of  $\text{LCA}_T(i, j)$  which is included in this sum and is  $\frac{1}{4^{d_T(\text{LCA}_T(i,j))}}$ . Thus we can bound  $\lg \frac{\sum_{k=\min(i,j)}^{\max(i,j)} w_k^T}{\min(w_i^T, w_j^T)}$  as follows:  $\lg \frac{\sum_{k=\min(i,j)}^{\max(i,j)} w_k^T}{\min(w_i^T, w_j^T)} \geq \lg \frac{\frac{1}{4^{d_T(\text{LCA}_T(i,j))}}}{\min\left(\frac{1}{4^{d_T(i)}}, \frac{1}{4^{d_T(j)}}\right)} = 2 \max(d_T(i), d_T(j)) - 2d_T(\text{LCA}_T(i, j)) \geq d_T(i) + d_T(j) - 2d_T(\text{LCA}_T(i, j)) = r(T, i, j)$

Similarly, an upper bound on  $\sum_{k=\min(i,j)}^{\max(i,j)} w_k^T$  is twice the weight of  $\text{LCA}_T(i, j)$ :  $\frac{2}{4^{d_T(\text{LCA}_T(i,j))}}$ . This is because each of the two paths down from the LCA have weights that when summed telescope to less than half that of the LCA:  $\lg \frac{\sum_{k=\min(i,j)}^{\max(i,j)} w_k^T}{\min(w_i^T, w_j^T)} \leq \lg \frac{\frac{2}{4^{d_T(\text{LCA}_T(i,j))}}}{\min\left(\frac{1}{4^{d_T(i)}}, \frac{1}{4^{d_T(j)}}\right)} = 2 \max(d_T(i), d_T(j)) - 4d_T(\text{LCA}_T(i, j)) \leq 2d_T(i) + 2d_T(j) - 4d_T(\text{LCA}_T(i, j)) = 2r(T, i, j)$ . □

### 4 Proof of Main Theorem

Here we combine the results of the previous two sections to show that the runtime of the optimal tree with lazy finger is asymptotically the weighted dynamic finger bound for the best choice of weights.

**Theorem 2**

$$\min_T \left\{ \sum_{i=1}^m r(T, x_{i-1}, x_i) \right\} = \Theta \left( \min_W \left\{ \sum_{i=1}^m \lg \frac{\sum_{k=\min(x_i, x_{i-1})}^{\max(x_i, x_{i-1})} w_k}{\min(w_{x_{i-1}}, w_{x_i})} \right\} \right)$$



*Proof.* Start by setting  $T^{\min}$ , to be the optimal tree. That is,  $T^{\min} = \operatorname{argmin}_T \{\sum_{i=1}^m r(T, x_{i-1}, x_i)\}$ :

$$\min_T \{\sum_{i=1}^m r(T, x_{i-1}, x_i)\} = \sum_{i=1}^m r(T^{\min}, x_{i-1}, x_i)$$

Using Lemma 1 there is a constant  $c$  such that:

$$\geq c \sum_{i=1}^m \lg \frac{\sum_{k=\min(x_i, x_{i-1})}^{\max(x_i, x_{i-1})} w_k^{T^{\min}}}{\min(w_{x_{i-1}}^{T^{\min}}, w_{x_i})}$$

The weights  $w^{T^{\min}}$  are a lower bound on the sum with the optimal weights

$$\geq c \min_W \sum_{i=1}^m \lg \frac{\sum_{k=\min(x_i, x_{i-1})}^{\max(x_i, x_{i-1})} w_k}{\min(w_{x_{i-1}}, w_{x_i})}$$

Using Theorem 1, there is a constant  $c'$  such that:

$$\geq c' \sum_{i=1}^m r(T_w, x_{i-1}, x_i)$$

The sum with  $T_w$  is at most the sum for optimal  $T$ :

$$\geq \min_T \{\sum_{i=1}^m r(T, x_{i-1}, x_i)\}$$

□

## 5 Hierarchy and Limitations of Models

In this section we show there is a strict hierarchy of runtimes from the root finger static BST model to the lazy finger static BST model to the rotation-based BST model. Let  $OPT(X)$  be the fastest any binary search with rotations can execute  $X$ .

**Theorem 3.** *For any sequence  $X$ ,  $\min_T R_{root}(T, X) = \Omega(\min_T R_{lazy}(T, X)) = \Omega(OPT(X))$ . Furthermore there exist classes of search sequences of any length  $m$ ,  $X'_m$  and  $X''_m$  such that  $\min_T R_{root}(T, X'_m) = \omega(\min_T R_{lazy}(T, X'_m))$  and  $\min_T R_{lazy}(T, X''_m) = \omega(OPT(X''_m))$ .*

*Proof.* We address each of the claims of this theorem separately.

*Root finger can be simulated with lazy finger:*  $\min_T R_{root}(T, X) = \Omega(\min_T R_{lazy}(T, X))$ . For lazy finger, moving up to the LCA and back down is no more work than than moving to the root and back down, which is exactly the double of the cost of the root finger method.

*Lazy finger can be simulated with a rotation-based tree:*  $\min_T R_{lazy}(T, X) = \Omega(OPT(X))$ . The normal definition of a tree allowing rotations has a finger that starts at the root at every operation and can move around the tree performing rotations. The work of Demaine et al. [10] shows how to simulate with constant-factor overhead any number of lazy fingers in a tree that allows rotations in the normal tree with rotations and one single pointer that starts at the root. This transformation can be used on a static tree with lazy finger to get the result.

*Some sequences can be executed quickly with lazy finger but not with root finger:* There is a  $X'_m$  such that  $\min_T R_{root}(T, X'_m) = \omega(\min_T R_{lazy}(T, X'_m))$ .

One choice of  $X'_m$  is the sequential search sequence  $1, 2, \dots, n, 1, 2, \dots$  repeated until a search sequence of length  $m$  is created. So long as  $m \geq n$ , this takes time  $O(m)$  to execute on any tree using lazy finger, but takes  $\Omega(m \lg n)$  time to execute on every tree using root finger.

*Some sequences can be executed quickly using a BST with rotations, but not with lazy finger.* Pick some small  $k$ , say  $k = \lg n$ . Create the sequence  $X''_m$  in rounds as follows: In each round pick  $k$  random elements from  $1..n$ , search each of them once, and then perform  $n$  random searches on these  $k$  elements. Continue with more rounds until a total of  $m$  searches are performed. A splay tree can perform this in time  $O(m \lg k)$ . This is because splay trees have the working-set bound, which states that the amortized time to search an item is at most big-O of the logarithm of the number of different things searched since the last time that item was searched. For the sequence  $X''_m$ , the  $n$  random searches in each round have been constructed to have a working set bound of  $O(\lg k)$  amortized, while the  $k$  other searches in each round have a working set bound of  $O(\lg n)$  amortized. Thus the total cost to execute  $X''_m$  on a splay tree is  $O\left(\frac{m}{n+k}(n \lg k + k \lg n)\right)$  which is  $O(m \lg \lg n)$  since  $k = \lg n$ .

However, for a static tree with lazy finger,  $X''_m$  is basically indistinguishable from a random sequence and takes  $\Omega(m \lg n)$  expected time. This is because the majority of the searches are random searches where the previous item was a random search, and in any static tree the expected distance between two random items is  $\Omega(\lg n)$ .  $\square$

**Lemma 2.** *The runtime of a BST in any model cannot be related to the conditional entropy of the search sequence.*

*Proof.* Wilber [20] proved that the bit reversal sequence is performed in  $\Omega(n \lg n)$  time in an optimal dynamic BST. This sequence is a precise permutation of all elements in the tree. However, any single permutation repeated over and over has a conditional entropy of 0, since every search is completely determined by the previous one.  $\square$

## 6 Constructing the Optimal Lazy Finger BST

Recall that  $f_{a,b} = f_X(a,b)$  is the number of searches in  $X$  where the current search is to  $b$  and the previous search is to  $a$ , and  $f_X(a)$  is the number of searches to  $a$  in  $X$ . We will first describe one method to compute the cost to execute  $X$  on some tree  $T$ . Suppose the nodes in  $[a,b]$  constitute the nodes of some subtree of  $T$ , call it  $T_{a,b}$  and denote the root of the subtree as  $r(T_{a,b})$ . We now present a recursive formula for computing the expected cost of a single search in  $T$ . Let  $R_{\text{lazy}}(T, X, a, b)$  be the number of edges traversed in  $T_{a,b}$  when executing  $X$ . Thus,  $R_{\text{lazy}}(T, X, 1, n)$  equals the runtime  $R_{\text{lazy}}(T, X)$ . There is a recursive formula for  $R_{\text{lazy}}(T, X, a, b)$ :

$$R_{\text{lazy}}(T, X, a, b) = \begin{cases} 0 & \text{if } b < a \\ \overbrace{R_{\text{lazy}}(T, X, a, r(T_{a,b}) - 1)}^{(a)} \\ \quad + \overbrace{R_{\text{lazy}}(T, X, r(T_{a,b}) + 1, b)}^{(b)} \\ \quad + 2 \sum_{\substack{i \in [a, r(T_{a,b}) - 1] \\ j \in [r(T_{a,b}) + 1, b]}}^{(c)} (f_{i,j} + f_{j,i}) \\ \quad + \overbrace{\sum_{i \neq r} (f_{i,r(T_{a,b})} + f_{r(T_{a,b}),i})}^{(d)} \\ \quad + \overbrace{\sum_{\substack{i \in [a,b] \\ i \neq r(T_{a,b}) \\ j \notin [a,b]}}^{(e)} (f_{i,j} + f_{j,i})} & \text{otherwise} \end{cases}$$

The formula is long but straightforward. First we recursively include the number of edges traversed in the left (a) and right (b) subtrees of the root  $r(T_{a,b})$ . Thus, all that is left to account for is traversing the edges between the root of the subtree and its up to two children. Both edges to its children are traversed when a search moves from the left to right subtree of  $r_{a,b}$  or vice-versa (c). A single edge to a child of the  $r(T_{a,b})$  traversed if a search moves from either the left or right subtrees of  $r(T_{a,b})$  to  $r(T_{a,b})$  itself or vice-versa (d), or if a search moves from any node but the root in the current subtree containing the nodes  $[a, b]$  out to the rest of  $T$  or vice-versa (e).

This formula can easily be adjusted into one to determine the optimal cost over all trees—since at each step the only dependence on the tree was is root of the current subtree, the minimum can be obtained by trying all possible roots. Here is the resultant recursive formulation for the minimum number of edges traversed in and among all subtrees containing  $[a, b]$ :

$$\min_T R_{\text{lazy}}(T, X, a, b) = \begin{cases} 0 & \text{if } b < a \\ \min_{r \in [a,b]} \left\{ \begin{array}{l} \min_T R_{\text{lazy}}(T, X, a, r - 1) \\ + \min_T R_{\text{lazy}}(T, X, r + 1, b) \\ + 2 \sum_{\substack{i \in [a, r - 1] \\ j \in [r + 1, b]}} (f_{i,j} + f_{j,i}) \\ + \sum_{i \neq r} (f_{i,r} + f_{r,i}) \\ + \sum_{\substack{i \in [a,b] \\ i \neq r \\ j \notin [a,b]}} (f_{i,j} + f_{j,i}) \end{array} \right\} & \text{otherwise} \end{cases}$$

This formula can trivially be evaluated using dynamic programming in  $O(n^5)$  time as there are  $O(n^3)$  choices for  $a, b$ , and  $r$  and evaluating the summations in the brute-force way takes time  $O(n^2)$ . The dynamic programming gives not

only the cost of the best tree, but the minimum roots chosen at each step gives the tree itself. The runtime can be improved to  $O(n^3)$  by observing that when  $f$  is viewed as a 2-D array, each of the sums is simply a constant number of partial sum queries on the array  $f$ , each of which can be answered in  $O(1)$  time after  $O(n^2)$  preprocessing. (The folklore method of doing this is to store all the 2-D partial sums from the origin; a generic partial sum can be computed from these with a constant number of additions and subtractions).

We summarize this result in the following theorem:

**Theorem 4.** *Given the pairwise frequencies  $f_X$  finding the tree that minimizes the execution time of search sequence  $X$  using lazy finger takes time  $O(n^3)$ .*

This algorithm computes an optimal tree, and takes time linear in the size of the frequency table  $f$ . Computing  $f$  from  $X$  can be done in  $O(m)$  time, for a total runtime of  $O(m + n^3)$ . It remains open if there is any approach to speed up the computation of the optimal tree, or an approximation thereof. Note that although our closed form expression of the asymptotic runtime of the best tree was stated in terms of an optimal choice of weights, the dynamic program presented here in no way attempts to compute these weights. It would be interesting if some weight-based method were to be discovered.

## 7 Multiple Trees Structure

Here we present a static data structure in the comparison model on a pointer machine that guarantees an average search time of  $O(H_c(f_X) \log_d n)$  for any fixed value  $1 \leq d \leq n$ , a runtime which we have shown to be impossible for any BST algorithm, static or dynamic. This data structure requires  $O(dn)$  space. In particular, setting  $d = n^\epsilon$  gives a search time of  $O(H_c(f_X))$  with space  $O(n^{1+\epsilon})$  for any  $\epsilon > 0$ . The purpose of this structure is to demonstrate that while no tree can have a runtime related to the conditional entropy, pointer based structures can.

As a first attempt, a structure could be made of  $n$  binary search trees  $T_1, T_2, \dots, T_n$  where each tree  $T_i$  is an optimal static tree given the previous search was to  $i$ . By using tree  $T_{x_{i-1}}$  to execute search  $T_i$ , the asymptotic conditional entropy can be easily obtained. However the space of this structure is  $O(n^2)$ . Thus space can be reduced by observing the nodes not near the root of every tree are being executed slowly and thus need not be stored in every tree.

The *multiple trees structure* has two main parts. It is composed first by a complete binary search tree  $T'$  containing all of  $S$ . Thus the height of  $T'$  is  $O(\lg n)$ . The second part is  $n$  binary search trees  $\{T_1, T_2, \dots, T_n\}$ . A tree  $T_i$  contains the  $d$  elements  $j$  that have the greatest frequencies  $f_X(i, j)$ ; these are the  $j$  elements most frequently searched after that  $i$  has been searched. The depth of an element  $j$  in  $T_i$  is  $O(\lg \frac{f_X(i)}{f_X(i, j)})$ . For each element  $j$  in the entire structure we add a pointer linking  $j$  to the root of  $T_j$ . The tree  $T'$  uses  $O(n)$  space and every tree  $T_j$  uses  $O(d)$  space. Thus the space used by the entire structure is  $O(dn)$ .

Suppose we have just searched the element  $i$  and our finger search is located on the root of  $T_i$ . Now we proceed to the next search to the element  $j$  in the following way: Search  $j$  in  $T_i$ . If  $j$  is in  $T_i$  then we are done, otherwise search  $j$  in  $T'$ . After we found  $j$  either in  $T_j$  or  $T'$  we move the finger to the root of  $T_j$  by following the aforementioned pointer.

If  $j$  is in  $T_i$  then it is found in time  $O(\lg \frac{f_X(i)}{f_X(i,j)})$ . Otherwise if  $y$  is found in  $T'$ , then it is found in  $O(\lg n)$  time. We know that if  $y$  is not in  $T_x$  this means that optimally it requires  $\Omega(\lg d)$  comparisons to be found since  $T_x$  contains the  $d$  elements that have the greatest probability to be searched after that  $x$  has been accessed. Hence every search is at most  $O(\lg n / \lg d)$  times the optimal search time of  $O(\lg \frac{f_X(i)}{f_X(i,j)})$ . Thus a search for  $x_i$  in  $X$  takes time  $O\left(\log_d n \lg \frac{f_X(x_i)}{f_X(x_{i-1}, x_i)}\right)$ . Summing this up over all  $m$  searches  $x_i$  in  $X$  gives the runtime to execute  $X$ :

$$\begin{aligned} O\left(\sum_{i=1}^m \log_d n \lg \frac{f_X(x_i)}{f_X(x_{i-1}, x_i)}\right) &= O\left(\sum_{a=1}^n \sum_{b=1}^n f_X(a, b) \log_d n \lg \frac{f_X(a)}{f_X(x_a, x_b)}\right) \\ &= O(mH_c(f_X) \log_d n) \end{aligned}$$

We summarize this result in the following theorem:

**Theorem 5.** *Given the pairwise frequencies  $f_X$  and a constant  $d$ ,  $1 \leq d \leq n$ , the multiple trees structure executes  $X$  in time  $O(mH_c(f_X) \log_d n)$  and uses space  $O(nd)$ .*

We conjecture that no pointer-model structure has space  $O(n)$  and search cost  $O(H_c(f_X))$ .

## References

1. Adelson-Velskij, G.M., Landis, E.M.: An Algorithm for the Organization of Information. *Doklady Akademii Nauk USSR* **146**(2), 263–266 (1962)
2. Alon, N., Spencer, J.: *The Probabilistic Method*. John Wiley (1992)
3. Aragon, C.R., Seidel, R.: Randomized search trees. In: FOCS, pp. 540–545. IEEE Computer Society (1989)
4. Bender, M.A., Duan, Z., Iacono, J., Jing, W.: A locality-preserving cache-oblivious dynamic dictionary. *J. Algorithms* **53**(2), 115–136 (2004)
5. Bronson, N.G., Casper, J., Chafi, H., Olukotun, K.: A practical concurrent binary search tree. In: Govindarajan, R., Padua, D.A., Hall, M.W. (eds.) PPOPP, pp. 257–268. ACM (2010)
6. Clark, D.R., Ian Munro, J.: Efficient suffix trees on secondary storage (extended abstract). In: Tardos, É. (ed.) SODA, pp. 383–391. ACM/SIAM (1996)
7. Cole, R.: On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM J. Comput.* **30**(1), 44–85 (2000)
8. Cole, R., Mishra, B., Schmidt, J.P., Siegel, A.: On the dynamic finger conjecture for splay trees. part i: Splay sorting log n-block sequences. *SIAM J. Comput.* **30**(1), 1–43 (2000)
9. Demaine, E.D., Iacono, J., Langerman, S.: Worst-case optimal tree layout in a memory hierarchy. CoRR, cs.DS/0410048 (2004)

10. Demaine, E.D., Iacono, J., Langerman, S., Özkan, Ö.: Combining binary search trees. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013, Part I. LNCS, vol. 7965, pp. 388–399. Springer, Heidelberg (2013)
11. Fredman, M.L.: Two applications of a probabilistic search technique: Sorting  $x + y$  and building balanced search trees. In: Rounds, W.C., Martin, N., Carlyle, J.W., Harrison, M.A. (eds.) STOC, pp. 240–244. ACM (1975)
12. Gil, J., Itai, A.: How to pack trees. *J. Algorithms* **32**(2), 108–132 (1999)
13. Guibas, L.J., Sedgwick, R.: A dichromatic framework for balanced trees. In: FOCS, pp. 8–21. IEEE Computer Society (1978)
14. Iacono, J., Mulzer, W.: A static optimality transformation with applications to planar point location. *Int. J. Comput. Geometry Appl.* **22**(4), 327–340 (2012)
15. Knuth, D.E.: Optimum binary search trees. *Acta Inf.* **1**, 14–25 (1971)
16. Mehlhorn, K.: Nearly optimal binary search trees. *Acta Inf.* **5**, 287–295 (1975)
17. Seidel, R., Aragon, C.R.: Randomized search trees. *Algorithmica* **16**(4/5), 464–497 (1996)
18. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. *J. ACM* **32**(3), 652–686 (1985)
19. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.* **6**(3), 80–82 (1977)
20. Wilber, R.E.: Lower bounds for accessing binary search trees with rotations. *SIAM J. Comput.* **18**(1), 56–67 (1989)

# **Fixed-Parameter Tractable Algorithms I**

# Minimum-Cost $b$ -Edge Dominating Sets on Trees

Takehiro Ito<sup>1</sup>, Naonori Kakimura<sup>2</sup>, Naoyuki Kamiyama<sup>3</sup>(✉),  
Yusuke Kobayashi<sup>2</sup>, and Yoshio Okamoto<sup>4</sup>

<sup>1</sup> Tohoku University, Sendai, Japan  
takehiro@ecei.tohoku.ac.jp

<sup>2</sup> University of Tokyo, Tokyo, Japan  
kakimura@global.c.u-tokyo.ac.jp, kobayashi@mist.i.u-tokyo.ac.jp

<sup>3</sup> Kyushu University, Fukuoka, Japan  
kamiyama@imi.kyushu-u.ac.jp

<sup>4</sup> University of Electro-Communications, Tokyo, Japan  
okamoto@uec.ac.jp

**Abstract.** We consider the minimum-cost  $b$ -edge dominating set problem. This is a generalization of the edge dominating set problem, but the computational complexity for trees is an astonishing open problem. We make steps toward the resolution of this open problem in the following three directions. (1) We give the first combinatorial polynomial-time algorithm for paths. Prior to our work, the polynomial-time algorithm for paths used linear programming, and it was known that the linear-programming approach could not be extended to trees. Thus, our algorithm would yield an alternative approach to a possible polynomial-time algorithm for trees. (2) We give a fixed-parameter algorithm for trees with the number of leaves as a parameter. Thus, a possible **NP**-hardness proof for trees should make use of trees with unbounded number of leaves. (3) We give a fully polynomial-time approximation scheme for trees. Prior to our work, the best known approximation factor was two. If the problem is **NP**-hard, then a possible proof cannot be done via a gap-preserving reduction from any **APX**-hard problem unless **P** = **NP**.

## 1 Introduction

Covering problems are very fundamental in the study of graph algorithms. By objects to cover and objects to be covered, the following terms are assigned. When vertices cover vertices, we obtain the minimum dominating set problem;

---

Takehiro Ito: Supported by JSPS KAKENHI Grant Numbers 25106504, 25330003.

Naonori Kakimura: Supported by JST, ERATO, Kawarabayashi Large Graph Project, and by JSPS KAKENHI Grant Numbers 25730001, 24106002.

Naoyuki Kamiyama: Supported by JSPS KAKENHI Grant Number 24106005.

Yusuke Kobayashi: Supported by JST, ERATO, Kawarabayashi Large Graph Project, and by JSPS KAKENHI Grant Numbers 24106002, 24700004.

Yoshio Okamoto: Supported by Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science (JSPS).



when vertices cover edges, we obtain the minimum vertex cover problem; when edges cover vertices, we obtain the minimum edge cover problem; when edges cover edges, we obtain the minimum edge dominating set problem. While the minimum edge cover problem can be solved in polynomial time, the other three problems are **NP**-hard for general graphs. Nevertheless, they can be solved in linear time for trees.

In some applications, variations of these problems are considered. One variation imposes cost on objects to cover, and another variation imposes demand on objects to be covered. Then, we want to select some objects to cover, possibly multiple times, so that each object to be covered is covered at least as many times as its demand. The objective is to minimize the total cost of selected objects multiplied by the times they are selected. The focus of this paper is the version for the minimum edge dominating set problem.

Formally, we define the minimum-cost  $b$ -edge dominating set problem ( $b$ -EDS, for short) as follows. In  $b$ -EDS, we are given a simple graph  $G$ . We denote by  $VG$  and  $EG$  the sets of vertices and edges of  $G$ , respectively. We adopt this notation for any graphs in this paper. In addition, a demand function  $b: EG \rightarrow \mathbb{Z}_+$  and a cost function  $c: EG \rightarrow \mathbb{R}_+$  are given.<sup>1</sup> For each edge  $e$  of  $G$ , we denote  $\delta(e)$  by the set of all edges sharing end vertices with  $e$ , including  $e$  itself. For each vertex  $v$  of  $G$ , we denote by  $\delta(v)$  the set of edges containing  $v$ . A vector  $s$  in  $\mathbb{Z}_+^{EG}$  is called a  $b$ -edge dominating vector of  $G$ , if  $s(\delta(e)) \geq b(e)$  for every edge  $e$  of  $G$ .<sup>2</sup> The value  $s(e)$  represents the number of times the edge  $e$  is selected. The cost of a  $b$ -edge dominating vector  $s$  of  $G$ , denoted by  $\text{cost}(s)$ , is defined as  $\langle c, s \rangle$ , where  $\langle \cdot, \cdot \rangle$  represents the inner product of two vectors. The goal of  $b$ -EDS is to find a  $b$ -edge dominating vector with the minimum cost. It is known [13] that  $b$ -EDS is **NP**-hard for general graphs even if  $b(e) = 1$  and  $c(e) = 1$  for every edge  $e$  of  $G$ . An  $8/3$ -approximation algorithm for general graphs and a 2-approximation algorithm for bipartite graphs were presented [1]. When  $b(e) = 1$  for every edge  $e$  of  $G$ , 2-approximation algorithms were also proposed [6, 9].

One astonishing open problem on  $b$ -EDS is to determine the computational complexity of the problem on trees. It is not known if the problem is **NP**-hard or polynomial-time solvable. If  $b(e) = 1$  for every edge  $e$  of  $G$  or  $c(e) = 1$  for every edge  $e$  of  $G$ , then  $b$ -EDS can be solved in polynomial time for trees [2].<sup>3</sup> Therefore, the combination of arbitrary  $b$  and arbitrary  $c$  makes the problem troublesome. The best known approximation factor is two, which is a consequence from bipartiteness of trees [1]. Even for paths, no combinatorial polynomial-time algorithm was known while a strongly polynomial-time algorithm can be designed via linear programming [10, 11].

<sup>1</sup> We denote by  $\mathbb{Z}$ ,  $\mathbb{Z}_+$ ,  $\mathbb{R}$ , and  $\mathbb{R}_+$  the sets of integers, nonnegative integers, real numbers, and nonnegative real numbers, respectively.

<sup>2</sup> For each set  $X$ , each vector  $\xi$  in  $\mathbb{R}^X$ , and each subset  $Y$  of  $X$ , we define  $\xi(Y) := \sum_{x \in Y} \xi(x)$ . Thus,  $s(\delta(e)) = \sum_{e' \in \delta(e)} s(e')$ .

<sup>3</sup> In [2], the authors claimed that  $b$ -EDS on trees can be solved in polynomial time via linear programming. However this claim is not correct (see [3]).

## 1.1 Contributions and Techniques

We make steps toward the resolution of the complexity status of  $b$ -EDS on trees in the following three directions.

**(1) Combinatorial Algorithm for Paths.** In Section 2, we give the first combinatorial algorithm for  $b$ -EDS on paths which runs in strongly polynomial time. This result would yield an alternative approach to a possible polynomial-time algorithm for trees for the attempt to prove that  $b$ -EDS can be solved in polynomial time. A polynomial-time algorithm using linear programming could not be extended to trees because the coefficient matrix of a natural integer-programming formulation of the problem is totally unimodular for paths, but not necessarily so for trees.

To give a combinatorial strongly polynomial-time algorithm for paths, we first give a dynamic-programming algorithm which runs in pseudo-polynomial time. Then, we construct a “compact” representation of the DP table. To construct such a compact representation of the DP table, we find a partition  $\mathcal{R}$  of the domain of the DP table so that in each part in  $\mathcal{R}$ , values of the DP table can be represented by a linear function. We will prove that we can construct such a partition whose size is bounded by a polynomial in the input size, which implies that we can “simulate” our dynamic-programming algorithm in strongly polynomial time. To the best of the authors’ knowledge, this technique of compressing the DP table is new, and should be of independent interest.

**(2) Fixed-Parameter Algorithm for Trees.** In Section 3, we give a fixed-parameter algorithm for  $b$ -EDS on trees with the number of leaves as a parameter. This result implies that when the number of leaves is constant, then the problem can be solved in polynomial time. Therefore, if we want to prove that the problem is **NP**-hard, then a possible reduction should make use of trees with unbounded number of leaves.

To give a fixed-parameter algorithm for trees, the following fact plays an important role. On paths, there exists no gap between a natural integer programming formulation of  $b$ -EDS and its linear-programming relaxation. By using this fact, we will prove that  $b$ -EDS on trees with constant number of leaves can be formulated as a mixed integer program with constant number of integer variables. Thus, a fixed-parameter algorithm follows from the result of Lenstra [8] on fixed-parameter tractability of mixed integer programming when the number of integer variables is a parameter.

**(3) FPTAS for Trees.** In Section 4, we give a fully polynomial-time approximation scheme (FPTAS, for short) for  $b$ -EDS on trees. Prior to our work, the best known approximation factor was two [1]. Thus, our algorithm improves the approximation factor drastically. This result implies that  $b$ -EDS on trees is not strongly **NP**-hard unless **P** = **NP**. Furthermore, if we want to prove the problem

is **NP**-hard, then a possible proof cannot be done via a gap-preserving reduction from any **APX**-hard problem unless **P** = **NP**.

To give an FPTAS for trees, we generalize the problem. In this generalization, (i) there may exist parallel edges, (ii) each edge can be chosen at most once, and (iii) each edge has an “influence.” If we choose an edge  $e$  with influence  $p(e)$ , then each edge in  $\delta(e)$  is covered  $p(e)$  times by  $e$ . We first give a pseudo-polynomial-time algorithm for this generalized problem on multitrees. Then, we give a reduction from  $b$ -EDS on trees to this generalized problem on multitrees. Finally, we prove that a pseudo-polynomial-time algorithm for this generalized problem on multitrees yields an FPTAS for  $b$ -EDS on trees.

## 1.2 Preliminaries: Polynomial-Time Algorithms and NP-Hardness

For the problem with numerical inputs, several notions of polynomial-time algorithms appear in the literature. Here, we summarize them.

Consider a problem in which we are given a combinatorial object  $O$  (in our case, a tree  $T$ ) and a set of numbers (in our case  $b(e)$  and  $c(e)$  for all  $e \in ET$ ) at most  $M$ . Suppose that the object  $O$  has size  $n$  (in our case  $T$  has  $n$  vertices).

An algorithm runs in *strongly polynomial time* if the running time is bounded by a polynomial in  $n$ . It runs in *weakly polynomial time* if the running time is bounded by a polynomial in  $n$  and  $\log M$ . It runs in *pseudo-polynomial time* if the running time is bounded by a polynomial in  $n$  and  $M$ . It is easy to see that a strongly polynomial-time algorithm is a weakly polynomial-time algorithm, and a weakly polynomial-time algorithm is a pseudo-polynomial-time algorithm. In this context, a *polynomial-time algorithm* is a weakly polynomial-time algorithm.

The problem is *strongly NP-hard* if it is **NP**-hard even if  $M$  is bounded by a polynomial in  $n$ . The usual **NP**-hardness is referred to as *weakly NP-hardness*.

## 2 Combinatorial Algorithm for Paths

In this section, we present a combinatorial strongly polynomial-time algorithm for  $b$ -EDS on a path  $P$ . Let  $VP = \{v_1, v_2, \dots, v_n\}$ ,  $EP = \{e_1, e_2, \dots, e_{n-1}\}$ , and  $e_i = (v_i, v_{i+1})$  for each  $i \in \{1, 2, \dots, n-1\}$ . For each  $i \in \{1, 2, \dots, n\}$ , we denote by  $P_i$  the subpath of  $P$  induced by  $V_i = \{v_1, v_2, \dots, v_i\}$ . Let  $B = \max\{b(e) \mid e \in EP\}$ .

In the sequel, we first give a pseudo-polynomial-time algorithm for  $b$ -EDS on  $P$  (Sect. 2.1), and the time complexity will be improved to polynomial (Sect. 2.2).

### 2.1 Dynamic Programming

For each  $i \in \{2, 3, \dots, n\}$ , each  $x \in \{0, 1, \dots, B\}$ , and each  $y \in \{0, 1, \dots, 2B\}$ , define  $\rho_i(x, y)$  as the minimum cost of a vector  $s$  in  $\mathbb{Z}_+^{EP_i}$  satisfying (i)  $s(\delta(e_j)) \geq b(e_j)$  for every  $j \in \{1, 2, \dots, i-2\}$ , (ii)  $s(e_{i-1}) = x$ , and (iii)  $s(e_{i-2}) + y \geq b(e_{i-1})$ , where  $s(e_0) = 0$ . If such a vector  $s$  does not exist, then define  $\rho_i(x, y) = +\infty$ .

Roughly, these conditions mean that  $s$  satisfies the constraints for  $e_1, e_2, \dots, e_{i-1}$  if  $e_{i-1}$  is “chosen”  $x$  times and it is assumed to be “covered”  $s(e_{i-2}) + y$  times.

We can compute  $\rho_i(x, y)$  by using the following formulae. If  $i = 2$ , then (i)  $\rho_2(x, y) = c(e_1) \cdot x$  if  $y \geq b(e_1)$ , and (ii)  $\rho_2(x, y) = +\infty$  otherwise. If  $i \in \{3, 4, \dots, n\}$ , then

$$\rho_i(x, y) = c(e_{i-1}) \cdot x + \min\{\rho_{i-1}(\hat{x}, \hat{y}) \mid \hat{y} - \hat{x} = x, \hat{x} \geq b(e_{i-1}) - y\}.$$

Then, the value  $\rho_i(x, y)$  can be computed by looking at  $O(B)$  values of  $\rho_{i-1}$  for each  $i, x$ , and  $y$ . Since we have  $O(nB^2)$  choices of  $i, x$ , and  $y$ , we can compute all values of  $\rho_i(x, y)$  in  $O(nB^3)$  time. Since the optimal objective value of  $b$ -EDS on  $P$  is equal to  $\min\{\rho_n(x, y) \mid x = y\}$ , we can solve the problem in  $O(nB^3)$  time, which is pseudo-polynomial time.

### 2.2 Compression of the DP Table

Since the DP table  $\rho_i(x, y)$  has  $\Theta(B^2)$  entries for each  $i$ , explicit computation of the table does not yield a polynomial-time algorithm. To achieve polynomiality, we construct a compact representation of  $\rho_i$ . More precisely, we represent  $\rho_i$  as a piecewise linear function such that the number of regions is bounded by a polynomial in  $n$ .

For each function  $f: \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$ , define  $\text{dom } f = \{(x, y) \in \mathbb{R}^2 \mid f(x, y) < +\infty\}$ . A function  $f: \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$  is said to be *convex* if it is convex in  $\text{dom } f$ . A set  $\mathcal{R} = \{R_1, R_2, \dots, R_\ell\}$  of polygons (which might be unbounded) in  $\mathbb{R}^2$  is called a *partition* of  $\text{dom } f$  if (i)  $R_1 \cup R_2 \cup \dots \cup R_\ell = \text{dom } f$ , and (ii)  $R_i$  and  $R_j$  do not intersect except at their boundaries for every distinct  $i, j \in \{1, 2, \dots, \ell\}$ . For each  $i \in \{1, 2, \dots, \ell\}$  and each point  $(x, y)$  in  $R_i$ , suppose that a function  $f: \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$  satisfies the constraint  $f(x, y) = g_i(x, y)$ , where  $\mathcal{R} = \{R_1, R_2, \dots, R_\ell\}$  is a partition of  $\text{dom } f$  and  $g_i(x, y)$  is a linear function on  $\mathbb{R}^2$ . In such a case,  $f$  is said to be a *piecewise linear function with respect to  $\mathcal{R}$* .

Our algorithm is based on the following lemmas. Recall that the domain of  $\rho_i$  is  $\{0, 1, \dots, B\} \times \{0, 1, \dots, 2B\}$ . We now extend the domain of this function to  $\mathbb{R}^2$ . We define  $\bar{\rho}_2: \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$  by (i)  $\bar{\rho}_2(x, y) = c(e_1) \cdot x$  if  $y \geq b(e_1)$  and  $x \geq 0$ , and (ii)  $\bar{\rho}_2(x, y) = +\infty$  otherwise. For each  $i \in \{3, 4, \dots, n\}$ , define  $\bar{\rho}_i: \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$  by

$$\bar{\rho}_i(x, y) = c(e_{i-1}) \cdot x + \min\{\bar{\rho}_{i-1}(\hat{x}, \hat{y}) \mid \hat{y} - \hat{x} = x, \hat{x} \geq b(e_{i-1}) - y\}$$

if  $x \geq 0$ , and  $\bar{\rho}_i(x, y) = +\infty$  otherwise. The following lemmas reveal fundamental properties of  $\bar{\rho}_i$

- Lemma 1.** 1. For every  $i \in \{3, 4, \dots, n\}$ , we have  $\text{dom } \bar{\rho}_i = \{(x, y) \mid x \geq 0\}$ .  
 2. For every  $i \in \{2, 3, \dots, n\}$ , there exists a partition  $\mathcal{R}_i$  of  $\text{dom } \bar{\rho}_i$  such that  $\bar{\rho}_i$  is a convex piecewise linear function with respect to  $\mathcal{R}_i$  and  $|\mathcal{R}_i|$  is at most  $(3/2) \cdot i(i - 1)$ . Such a partition  $\mathcal{R}_i$  and a linear function on each part of  $\mathcal{R}_i$  can be computed in strongly polynomial time.

**Lemma 2.** *For every  $i \in \{2, 3, \dots, n\}$ , we have the following.*

1. *For every  $x \geq 0$  and every  $y_1, y_2 \geq B$ , we have  $\bar{\rho}_i(x, y_1) = \bar{\rho}_i(x, y_2)$ .*
2. *For every  $x \geq B$  and every  $y \in \mathbb{R}$ , we have  $\bar{\rho}_i(x, y) \geq \bar{\rho}_i(B, y)$ .*
3. *For every  $x \in \{0, 1, \dots, B\}$  and every  $y \in \{0, 1, \dots, 2B\}$ , we have  $\bar{\rho}_i(x, y) = \rho_i(x, y)$ .*

Assuming the lemmas, we may proceed as follows. By Lemma 1, we can compute in polynomial time  $\mathcal{R}_i$  and a linear function on each part  $R$  in  $\mathcal{R}_i$ . Since  $\bar{\rho}_i(x, y) = \rho_i(x, y)$  for every  $x \in \{0, 1, \dots, B\}$  and every  $y \in \{0, 1, \dots, 2B\}$  by Lemma 2, we can compute the optimal value of  $b$ -EDS on  $P$ , which is  $\min\{\rho_n(x, y) \mid x = y\}$ , in strongly polynomial time.

**Theorem 1.** *We can solve  $b$ -EDS on paths in strongly polynomial time.*

We are left with proofs of Lemmas 1 and 2. Due to the limitation of the space, we omit the full proofs. We sketch basic ideas for the proofs below.

### 2.3 Good Partitions and Zigzags

In the proof of Lemma 1, we stick to a partition of a special kind. Namely, a partition  $\mathcal{R}$  of  $\text{dom } f$  is called *good* if it satisfies the following.

For each part  $R$  in  $\mathcal{R}$ , there exist  $x_d, y_d, d_d$  in  $\mathbb{Z} \cup \{-\infty\}$  and  $x^u, y^u, d^u$  in  $\mathbb{Z} \cup \{+\infty\}$  such that  $R$  is a polygon defined by  $x_d \leq x \leq x^u$ ,  $y_d \leq y \leq y^u$ , and  $d_d \leq y - x \leq d^u$ .

The proof goes by induction on  $i$ . For fixed  $i \geq 4$ , assume that  $\text{dom } \bar{\rho}_{i-1} = \{(x, y) \mid x \geq 0\}$  and a good partition  $\mathcal{R}_{i-1}$  as in the statement has been obtained. By the definition of  $\bar{\rho}_i$ , for every point  $(x, y) \in \mathbb{R}^2$  with  $x \geq 0$ , we have

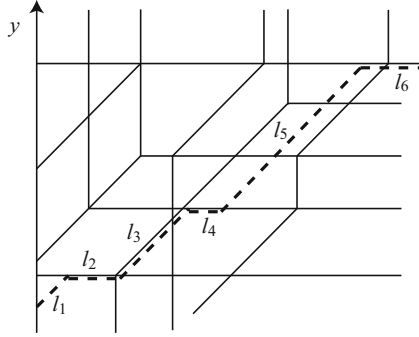
$$\begin{aligned} \bar{\rho}_i(x, y) &= c(e_{i-1}) \cdot x + \min\{\bar{\rho}_{i-1}(\hat{x}, \hat{y}) \mid \hat{y} - \hat{x} = x, \hat{x} \geq b(e_{i-1}) - y\} \\ &= c(e_{i-1}) \cdot x + \min_{\tilde{y} \leq y} \min\{\bar{\rho}_{i-1}(\hat{x}, \hat{y}) \mid \hat{y} - \hat{x} = x, \hat{x} = b(e_{i-1}) - \tilde{y}\} \\ &= c(e_{i-1}) \cdot x + \min_{\tilde{y} \leq y} \{\bar{\rho}_{i-1}(b(e_{i-1}) - \tilde{y}, b(e_{i-1}) - \tilde{y} + x)\}. \end{aligned}$$

We consider the functions  $g_1, g_2, g_3, g_4: \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$  defined by

$$\begin{aligned} g_1(x, y) &= \bar{\rho}_{i-1}(b(e_{i-1}) - y, b(e_{i-1}) - y + x), & g_2(x, y) &= \min_{\tilde{y} \leq y} g_1(x, \tilde{y}), \\ g_3(x, y) &= c(e_{i-1}) \cdot x + g_2(x, y), & g_4(x, y) &= \begin{cases} g_3(x, y) & \text{if } x \geq 0 \\ +\infty & \text{otherwise} \end{cases} \end{aligned}$$

for each point  $(x, y)$  in  $\mathbb{R}^2$ . Then, we have  $\bar{\rho}_i = g_4$ .

We take a careful look at the influence of  $g_1, g_2, g_3, g_4$  to the partition. The form of  $g_1$  suggests a linear transformation of the coordinate system, and indeed, by the definition of a good partition, such a transformation maps a good partition of  $\text{dom } \rho_{i-1}$  to a good partition of  $\text{dom } g_1$ . Assuming the existence of a good



**Fig. 1.** A good partition and an elementary zigzag of type (a)

partition for  $\text{dom } g_2$ , we can see that such a good partition is also a good partition of  $\text{dom } g_3$ , from which we can easily obtain a good partition of  $\text{dom } g_4$ . The size of partitions will not increase for those cases. Thus, the only problem may arise in finding a good partition of  $\text{dom } g_2$  from that of  $\text{dom } g_1$ . This case needs a special treatment, and gives rise to a concept of zigzags.

Let  $\mathcal{R} = \{R_1, R_2, \dots, R_\ell\}$  be a partition of  $\text{dom } f$ . The *boundary* of  $\mathcal{R}$  is the union of the boundaries of  $R_1, R_2, \dots, R_\ell$ . A *corner* of  $\mathcal{R}$  is a point in  $\mathbb{R}^2$  which is on the boundary of at least three regions in  $\mathcal{R} \cup \{\mathbb{R}^2 \setminus \text{dom } f\}$ .

For a good partition  $\mathcal{R}$  of  $\text{dom } f$ , an *elementary zigzag through  $\mathcal{R}$*  is a sequence  $l_1, l_2, \dots, l_t$  of line segments ( $l_1$  and  $l_t$  may be rays) in  $\text{dom } f$  such that  $l_1, l_2, \dots, l_t$  are oriented so that the head of  $l_i$  and the tail of  $l_{i+1}$  coincide for each  $i \in \{1, 2, \dots, t-1\}$ , and they satisfy one of the following conditions (see Fig. 1).

- a. The directions of  $l_i$  are  $(1, 1)$  and  $(1, 0)$ , alternately. If the direction of  $l_i$  is  $(1, 0)$ , then it is on the boundary of  $\mathcal{R}$ .
- b. The directions of  $l_i$  are  $(0, -1)$  and  $(-1, -1)$ , alternately. If the direction of  $l_i$  is  $(-1, -1)$ , then it is on the boundary of  $\mathcal{R}$ .
- c. The directions of  $l_i$  are  $(-1, 0)$  and  $(0, 1)$ , alternately. If the direction of  $l_i$  is  $(0, 1)$ , then it is on the boundary of  $\mathcal{R}$ .

Note that in the definition of elementary zigzags of type (a),  $l_i$  is not necessarily on the boundary of  $\mathcal{R}$  if its direction is  $(1, 1)$ . We also note that, elementary zigzags of types (b) and (c) are obtained from one of type (a) by linear transformation.

Our motivation for introducing elementary zigzags is the following lemma. We omit the proof.

**Lemma 3.** *Let  $f: \mathbb{R}^2 \rightarrow \mathbb{R} \cup \{+\infty\}$  be a convex piecewise linear function with respect to a good partition  $\mathcal{R}$  of  $\text{dom } f$ . Assume that, for any  $x \in \mathbb{R}$ , there exists a maximum minimizer  $h(x)$  (i.e., a maximum of the minimizers) of a 1-dimensional function  $f(x, \cdot)$ . Then,  $\{(x, h(x)) \mid x \in \mathbb{R}\}$  forms an elementary zigzag of type (a) on the boundary of  $\mathcal{R}$ .*

The construction of  $g_2$  from  $g_1$  can be done as follows. For fixed  $x \in \mathbb{R}$ , regard  $g_1(x, \cdot)$  as a 1-dimensional function, and define  $h(x)$  as the maximum minimizer of  $g_1(x, \cdot)$ . We may see that  $g_1$  satisfies the condition in Lemma 3. Thus,  $\{(x, h(x)) \mid x \in \mathbb{R}\}$  forms an elementary zigzag  $Z$  of type (a) on the boundary of a good partition of  $\text{dom } g_1$ . In particular,  $g_2(x, y) = g_1(x, y)$  if  $y \leq h(x)$  and  $g_2(x, y) = g_1(x, h(x))$  if  $y > h(x)$ . Therefore, the size of  $Z$  will contribute to the size of a good partition of  $\text{dom } g_2$ . This also means that we need to keep track of the size of such an elementary zigzag in the induction. To this end, we actually prove a stronger statement than Lemma 1, and use a more generalized concept.

### 3 Fixed-Parameter Algorithm for Trees

In this section, we give a fixed-parameter algorithm for  $b$ -EDS on trees with the number of leaves as a parameter. We first consider a subproblem on a path, which can be solved by linear programming, and then describe how to formulate  $b$ -EDS on trees using these subproblems.

#### 3.1 Subproblem

Let  $P$  be a path such that  $VP = \{v_1, v_2, \dots, v_n\}$ ,  $EP = \{e_1, e_2, \dots, e_{n-1}\}$ , and  $e_i = (v_i, v_{i+1})$  for each  $i \in \{1, 2, \dots, n-1\}$ . Given integers  $y(e_1), y(e_{n-1}), z(v_1)$ , and  $z(v_n)$ , we consider the problem of finding  $x$  in  $\mathbb{Z}_+^{EP}$  that minimizes  $\langle c, x \rangle$  subject to the following: when  $n \geq 3$ ,

$$\begin{aligned} x(e_{i-1}) + x(e_i) + x(e_{i+1}) &\geq b(e_i) \quad (\forall i \in \{2, 3, \dots, n-2\}), \\ x(e_2) + z(v_1) &\geq b(e_1), \quad x(e_{n-2}) + z(v_n) \geq b(e_{n-1}), \\ x(e_1) &= y(e_1), \quad x(e_{n-1}) = y(e_{n-1}), \end{aligned} \tag{1}$$

and, when  $n = 2$ ,

$$x(e_1) \leq z(v_1) + z(v_2) - b(e_1), \quad x(e_1) = y(e_1). \tag{2}$$

Note that these constraints mean that  $x(e_1) = y(e_1)$ ,  $x(e_{n-1}) = y(e_{n-1})$ , and if  $v_1$  and  $v_n$  are ‘‘covered’’ at least  $z(v_1)$  and  $z(v_n)$  times, respectively, then  $x$  is a feasible solution of  $b$ -EDS on  $P$ . This problem can be written as the following integer program:

$$\min\{\langle c, x \rangle \mid Ax \geq b', \ x \in \mathbb{Z}_+^{EP}\}, \tag{3}$$

where  $A$  is a 0-1 matrix and each entry of  $b'$  is integer if  $y(e_1), y(e_{n-1}), z(v_1)$ , and  $z(v_n)$  are integers. Since  $A$  is totally unimodular ( $A$  is an interval matrix) [10], the optimal value of the problem is equal to that of its linear-programming relaxation.

### 3.2 Algorithm

Let  $T$  be a given tree with at most  $\ell$  leaves. Let  $L$  be the set of all leaves, and let  $S$  be the set of all *hub vertices* of  $VT$  (i.e., vertices of degree at least three). Define  $F := \bigcup\{\delta(v) \mid v \in L \cup S\}$ . We denote by  $\mathcal{P}$  the set of all paths  $P$  such that both end vertices of  $P$  are in  $L \cup S$  and no inner vertex of  $P$  is in  $L \cup S$ . It is not difficult to see that  $\mathcal{P}$  gives a partition of  $ET$ . For each path  $P$  in  $\mathcal{P}$ , we denote by  $b_P$  the restriction of  $b$  to  $EP$ . We note that  $|L| \leq \ell$ ,  $|S| \leq \ell$ ,  $|\mathcal{P}| \leq 2\ell$ , and  $|F| \leq 2|\mathcal{P}| \leq 4\ell$ .

It is not difficult to see that  $b$ -EDS is equivalent to the problem of finding vectors  $x$  in  $\mathbb{Z}_+^{ET}$ ,  $y$  in  $\mathbb{Z}_+^F$ , and  $z$  in  $\mathbb{Z}_+^{L \cup S}$  that minimize  $\langle c, x \rangle$  subject to

$$\begin{aligned} z(v) &= y(\delta(v)) \text{ for each vertex } v \text{ in } L \cup S, \\ \text{Condition (1)} & \text{ for each path } P \text{ in } \mathcal{P} \text{ of length at least two,} \\ \text{Condition (2)} & \text{ for each path } P \text{ in } \mathcal{P} \text{ of length one.} \end{aligned} \tag{4}$$

By the arguments in the previous subsection, the optimal value does not change even if we drop the integrality of  $x$ . That is,  $b$ -EDS on  $T$  is equivalent to minimizing  $\langle c, x \rangle$  subject to  $x \in \mathbb{R}_+^{ET}$ ,  $y \in \mathbb{Z}_+^F$ ,  $z \in \mathbb{Z}_+^{L \cup S}$ , and (4), which is a mixed integer program. Indeed, suppose we have an optimal solution  $x$  in  $\mathbb{R}_+^{ET}$ ,  $y$  in  $\mathbb{Z}_+^F$ , and  $z$  in  $\mathbb{Z}_+^{L \cup S}$  for this mixed integer program. Then, for the fixed  $y$  in  $\mathbb{Z}_+^F$  and  $z$  in  $\mathbb{Z}_+^{L \cup S}$ , the restriction  $x_P$  of  $x$  on each path  $P$  in  $\mathcal{P}$  is an optimal solution to the linear-programming relaxation of (3), and hence there exists an integer optimal solution to the linear-programming relaxation because the coefficient matrix is totally unimodular. Since the number of integer variables in this mixed linear program is bounded by  $|F| + |L \cup S| \leq 6\ell$ , it can be solved in polynomial time by the following theorem (see also [10]).

**Theorem 2 (Lenstra [8]).** *The problem of solving  $\max\{\langle d_1, x \rangle + \langle d_2, y \rangle \mid A_1x + A_2y \leq b', x \text{ is integral}\}$ , where  $A_1$  has rank at most  $\ell$ , is fixed-parameter tractable with respect to  $\ell$ .*

Note that the current best running time is  $2^{O(\ell \log \ell)}$  times a polynomial of the input size [4, 5]. By using this theorem, we can obtain the following result.

**Theorem 3.** *The problem  $b$ -EDS is fixed-parameter tractable with respect to the number of leaves of an input tree.*

## 4 FPTAS for Trees

We denote by  $\text{OPT}(T)$  the optimal objective value for  $T$ . Precisely speaking,  $\text{OPT}(T)$  depends on  $b$  and  $c$ . However, we omit them for brevity. To give an FPTAS, we first introduce a generalization of  $b$ -EDS ( $b$ -GEDS, for short), and construct a pseudo-polynomial-time algorithm to solve  $b$ -GEDS. Based on the algorithm, we then give an FPTAS for  $b$ -EDS on trees.



In  $b$ -GEDS, an input graph  $G$  may have parallel edges. In addition to  $b(e)$  and  $c(e)$ , each edge  $e$  is associated with a nonnegative integer  $p(e)$ , called the *influence* of  $e$ . A subset  $D$  of  $E$  is called a *generalized  $b$ -edge dominating set* of  $G$ , if  $p(\delta(e) \cap D) \geq b(e)$  for every edge  $e$  of  $G$ . The *cost* of a generalized  $b$ -edge dominating set  $D$  of  $G$ , denoted by  $\text{cost}(D)$ , is defined as  $c(D)$ . The goal of  $b$ -GEDS is to find a generalized  $b$ -edge dominating set with the minimum cost; we denote by  $\text{GOPT}(G)$  the optimal objective value for a given multigraph  $G$ . We can prove that  $b$ -GEDS is weakly NP-hard on multitrees consisting of two vertices (we omit the proof). Notice that a graph  $T$  is called a *multitree* if it is a tree when we regard parallel edges as a single edge. We will show in Sect. 4.2 that  $b$ -EDS on trees can be reduced to  $b$ -GEDS on multitrees.

We note here that a pseudo-polynomial-time algorithm for  $b$ -EDS on trees can be obtained without resorting to  $b$ -GEDS. However, such an algorithm is not suited for applying the scale-and-round technique [7, 12] to obtain an FPTAS since bounding the error seems difficult; a similar situation occurs in the bounded knapsack problem [7, Chapter 7].

### 4.1 Pseudo-Polynomial-Time Algorithm for $b$ -GEDS on Multitrees

In this subsection, we prove the following theorem.

**Theorem 4.** *On a multitree  $T$  with  $m$  edges,  $b$ -GEDS can be solved in  $O(mB^4)$  time, where  $B := \max\{b(e) \mid e \in ET\}$ .*

*Proof.* We give an algorithm to solve  $b$ -GEDS in the desired time complexity. Our algorithm simply computes  $\text{GOPT}(T)$ . It is easy to modify our algorithm so that it finds a generalized  $b$ -edge dominating set with the minimum cost  $\text{GOPT}(T)$ .

We choose an arbitrary vertex  $r$  as the root of  $T$ , and regard  $T$  as a rooted tree. For each vertex  $v$  of  $T$ , let  $T_v$  be the subtree of  $T$  which is rooted at  $v$  and is induced by  $v$  and all descendants of  $v$  on  $T$ . Let  $w_1, w_2, \dots, w_q$  be the children of  $v$ , ordered arbitrarily. For each  $j \in \{1, 2, \dots, q\}$ , we denote by  $T_v^j$  the subtree of  $T$  induced by  $\{v\} \cup VT_{w_1} \cup VT_{w_2} \cup \dots \cup VT_{w_j}$ . For the sake of notational convenience, we denote by  $T_v^0$  the tree consisting of a single vertex  $v$ . Then,  $T_v = T_v^0$  for each leaf  $v$  of  $T$ . Let  $E_{vw_j}$  be the set of (multiple) edges joining  $v$  and  $w_j$ .

For each vertex  $v$  of  $T$ , each  $j \in \{0, 1, \dots, q\}$ , and each  $x, y \in \{0, 1, \dots, B\}$ , we define  $h_v^j(x, y)$  as the minimum cost of a subset  $D_v^j$  of  $ET_v^j$  subject to

1.  $p(\delta(e) \cap D_v^j) \geq b(e) \quad (\forall e \in ET_{w_k}, \forall k \in \{1, 2, \dots, j\})$ ,
2.  $p(\delta(v) \cap D_v^j) \geq x$ , and
3.  $y + p(\delta(e) \cap ET_{w_k} \cap D_v^j) \geq b(e) \quad (\forall e \in E_{vw_k}, \forall k \in \{1, 2, \dots, j\})$ .

If such a subset  $D_v^j$  does not exist, then define  $h_v^j(x, y) = +\infty$ . Roughly speaking,  $x$  guarantees the influence on  $v$  from  $\delta(v) \cap ET_v^j$ , and  $y$  specifies the total influence guaranteed around  $v$  in  $T$ . For the notational convenience, we sometimes denote  $h_v(x, y) = h_v^q(x, y)$ , where  $q$  is the number of children of  $v$ .

The proposed algorithm computes  $h_v^j(x, y)$  for each vertex  $v$  of  $T$ , each  $j \in \{0, 1, \dots, q\}$ , and all  $x, y \in \{0, 1, \dots, B\}$ , from the leaves of  $T$  to the root  $r$  of  $T$ , by means of dynamic programming. Since  $T_r = T$  for the root  $r$  of  $T$ , we can compute  $\text{GOPT}(T) = \min\{h_r(x, y) \mid x = y \in \{0, 1, \dots, B\}\}$  for a given multitree  $T$ . Therefore,  $\text{GOPT}(T)$  can be computed in  $O(B)$  time if we have computed the values  $h_r(x, y)$  for all  $x, y \in \{0, 1, \dots, B\}$ .

We now explain how to compute  $h_v^j(x, y)$ . We first consider the case where  $j = 0$ . Recall that, for each vertex  $v$  of  $T$ , the subtree  $T_v^0$  is defined to be a single vertex  $v$ . As the initialization, for each  $x, y \in \{0, 1, \dots, B\}$ , define  $h_v^0(x, y) = 0$ . This can be done in  $O(nB^2)$  time for all vertices of  $T$  and all  $x, y \in \{0, 1, \dots, B\}$ .

We then consider the case where  $j \geq 1$ , and hence  $v$  is an internal vertex of  $T$ . Suppose that we have already computed  $h_v^{j-1}(x, y)$  and  $h_{w_j}(x, y)$  for all  $x, y \in \{0, 1, \dots, B\}$ . For each  $z \in \{0, 1, \dots, B\}$ , let

$$\pi(E_{vw_j}, z) = \min\{c(D) \mid D \subseteq E_{vw_j}, p(D) \geq z\}.$$

If such a subset  $D$  does not exist, then we define  $\pi(E_{vw_j}, z) = +\infty$ . By a simple dynamic-programming algorithm similar to the knapsack problem [7], we can compute  $\pi(E_{vw_j}, z)$  for all  $z \in \{0, 1, \dots, B\}$  in  $O(|E_{vw_j}|B)$  time.

We now compute  $h_v^j(x, y)$  for each  $x, y \in \{0, 1, \dots, B\}$ . We first fix the influence  $z \in \{0, 1, \dots, x\}$  obtained by the choice of  $E_{vw_j}$ , and compute the following value  $h_v^j(x, y; z)$ :

$$h_v^j(x, y; z) = \pi(E_{vw_j}, z) + \min\{h_v^{j-1}(x - z, y) + h_{w_j}(x', x' + z)\}, \quad (5)$$

where the minimum above is taken over all  $x' \in \{0, 1, \dots, B\}$  such that  $y + x' \geq \max\{b(e) \mid e \in E_{vw_j}\}$ ; let  $h_v^j(x, y; z) = +\infty$  if such an integer  $x'$  does not exist. Then, we have

$$h_v^j(x, y) = \min\{h_v^j(x, y; z) \mid z \in \{0, 1, \dots, x\}\}. \quad (6)$$

We estimate the running time of this update computation. Recall that we can compute the values  $\pi(E_{vw_j}, z)$  for all  $z \in \{0, 1, \dots, B\}$  in  $O(|E_{vw_j}|B)$  time. For each  $x, y \in \{0, 1, \dots, B\}$ , by (5) and (6),  $h_v^j(x, y)$  can be computed in  $O(B^2)$  time. Thus, for each vertex  $v$  of  $T$  and each  $j \in \{1, 2, \dots, q\}$ , we can compute  $h_v^j(x, y)$  for all  $x, y \in \{0, 1, \dots, B\}$  in  $O(|E_{vw_j}|B + B^4)$  time. Then,  $h_r(x, y)$  for the root  $r$  can be computed in total time

$$\sum_{v \in VT} \sum_j O(|E_{vw_j}|B + B^4) = O(mB + nB^4) = O(mB^4),$$

where  $n = |VT|$  and  $m = |ET|$ . Notice that  $n \leq m + 1$  since  $T$  is a multitree. This completes the proof of Theorem 4.  $\square$

## 4.2 FPTAS for $b$ -EDS on Trees

By using the following proposition, we can reduce  $b$ -EDS to  $b$ -GEDS. We omit its proof.

**Proposition 1.** *Let  $(G', b', c')$  be an instance of  $b'$ -EDS with  $B' = \max\{b'(e) \mid e \in EG'\}$ . Then, one can construct an instance  $(G, b, c, p)$  of  $b$ -GEDS in polynomial time such that*

- a.  $|VG| = |VG'|$  and  $|EG| = |EG'| \cdot (\lceil \log B' \rceil + 1)$ ,
- b.  $\max\{b(e) \mid e \in EG\} = B'$ , and
- c. *there exists a  $b'$ -edge dominating vector  $s$  of  $G'$  with  $\text{cost}(s) = \gamma$  if and only if there exists a generalized  $b$ -edge dominating set  $D$  of  $G$  with  $\text{cost}(D) = \gamma$ .*

*In particular,  $\text{GOPT}(G) = \text{OPT}(G')$  holds.*

To obtain an FPTAS for  $b$ -EDS on trees, we now rephrase Theorem 4 when restricted to the instances corresponding to those of  $b$ -EDS. We omit the proof.

**Lemma 4.** *Let  $T$  be the multitree obtained by Proposition 1 from a tree  $T'$  of an instance of  $b'$ -EDS. Let  $U$  be any upper bound on  $\text{GOPT}(T)$ . Then,  $b$ -GEDS on  $T$  can be solved in  $O(mU^4)$  time, where  $m = |ET|$ .*

We finally give an FPTAS for  $b$ -EDS on trees, based on the pseudo-polynomial-time algorithm in Sect. 4.1.

**Theorem 5.** *There exists an FPTAS for  $b$ -EDS on trees.*

Our idea for the FPTAS is to combine techniques developed for the bounded knapsack problem [7, Chapter 7] and the scale-and-round technique [7, 12]. Due to the limitation of the space, we omit the detail.

**Acknowledgments.** The authors would like to thank Yusuke Matsumoto and Chien-Chung Huang for helpful discussions on this topic. The authors would like to thank anonymous referees for helpful comments.

## References

1. Berger, A., Fukunaga, T., Nagamochi, H., Parekh, O.: Approximability of the capacitated-edge dominating set problem. *Theor. Comput. Sci.* **385**(1–3), 202–213 (2007)
2. Berger, A., Parekh, O.: Linear time algorithms for generalized edge dominating set problems. *Algorithmica* **50**(2), 244–254 (2008)
3. Berger, A., Parekh, O.: Erratum to: Linear Time Algorithms for Generalized Edge Dominating Set Problems. *Algorithmica* **62**(1), 633–634 (2012)
4. Dadush, D., Peikert, C., Vempala, S.: Enumerative lattice algorithms in any norm via M-ellipsoid coverings. In: *FOCS*, pp. 580–589 (2011)
5. Dadush, D., Vempala, S.: Deterministic construction of an approximate M-ellipsoid and its applications to derandomizing lattice algorithms. In: *SODA*, pp. 1445–1456 (2012)
6. Fujito, T., Nagamochi, H.: A 2-approximation algorithm for the minimum weight edge dominating set problem. *Discrete Appl. Math.* **118**(3), 199–207 (2002)
7. Kellerer, H., Pferschy, U., Pisinger, D.: *Knapsack Problems*. Springer (2004)

8. Lenstra, H.W.: Integer programming with a fixed number of variables. *Math. Oper. Res.* **8**(4), 538–548 (1983)
9. Parekh, O.: Edge dominating and hypomatchable sets. In: *SODA*, 287–291 (2002)
10. Schrijver, A.: *Theory of Linear and Integer Programming*. John Wiley & Sons (1986)
11. Tardos, É.: A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research* **34**(2), 250–256 (1986)
12. Vazirani, V.V.: *Approximation algorithms*. Springer (2001)
13. Yannakakis, M., Gavril, F.: Edge dominating sets in graphs. *SIAM J. Appl. Math.* **38**(3), 364–372 (1980)

# Fixed-Parameter Tractability of Token Jumping on Planar Graphs

Takehiro Ito<sup>1</sup>(✉), Marcin Kamiński<sup>2</sup>, and Hiroataka Ono<sup>3</sup>

<sup>1</sup> Graduate School of Information Sciences, Tohoku University,  
Aoba-yama 6-6-05, Sendai 980-8579, Japan

takehiro@ecei.tohoku.ac.jp

<sup>2</sup> Department of Mathematics, Computer Science and Mechanics,  
University of Warsaw, Banacha 2, 02-097 Warsaw, Poland

mjk@mimuw.edu.pl

<sup>3</sup> Faculty of Economics, Kyushu University,  
Hakozaki 6-19-1, Higashi-ku, Fukuoka 812-8581, Japan

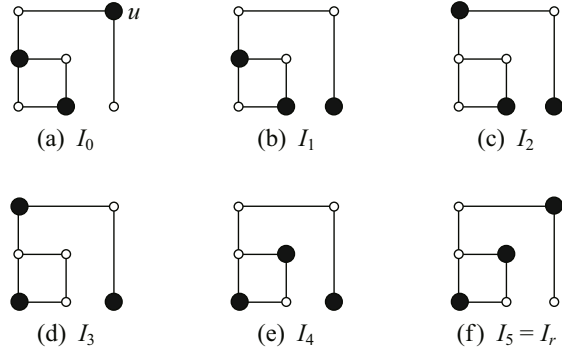
hirotaka@econ.kyushu-u.ac.jp

**Abstract.** Suppose that we are given two independent sets  $I_0$  and  $I_r$  of a graph such that  $|I_0| = |I_r|$ , and imagine that a token is placed on each vertex in  $I_0$ . The **TOKEN JUMPING** problem is to determine whether there exists a sequence of independent sets of the same cardinality which transforms  $I_0$  into  $I_r$  so that each independent set in the sequence results from the previous one by moving exactly one token to another vertex. This problem is known to be PSPACE-complete even for planar graphs of maximum degree three, and W[1]-hard for general graphs when parameterized by the number of tokens. In this paper, we present a fixed-parameter algorithm for **TOKEN JUMPING** on planar graphs, where the parameter is only the number of tokens. Furthermore, the algorithm can be modified so that it finds a shortest sequence for a yes-instance. The same scheme of the algorithms can be applied to a wider class of graphs which forbid a complete bipartite graph  $K_{3,t}$  as a subgraph for a fixed integer  $t \geq 3$ .

## 1 Introduction

The **TOKEN JUMPING** problem was introduced by Kamiński et al. [13], which can be seen as a “dynamic” version of independent sets in a graph. An *independent set* of a graph  $G$  is a set of vertices of  $G$  in which no two vertices are adjacent. (See Fig. 1, which depicts six different independent sets of the same graph.) Suppose that we are given two independent sets  $I_0$  and  $I_r$  of a graph  $G = (V, E)$  such that  $|I_0| = |I_r|$ , and imagine that a token is placed on every vertex in  $I_0$ . Then, the **TOKEN JUMPING** problem is to determine whether there exists a sequence  $\langle I_0, I_1, \dots, I_\ell \rangle$  of independent sets of  $G$  such that

- (a)  $I_\ell = I_r$ , and  $|I_0| = |I_1| = \dots = |I_\ell|$ ; and
- (b) for each index  $i \in \{1, 2, \dots, \ell\}$ ,  $I_i$  can be obtained from  $I_{i-1}$  by moving exactly one token on a vertex  $u \in I_{i-1}$  to another vertex  $v \notin I_{i-1}$ , and hence  $I_{i-1} \setminus I_i = \{u\}$  and  $I_i \setminus I_{i-1} = \{v\}$ .



**Fig. 1.** A sequence  $\langle I_0, I_1, \dots, I_5 \rangle$  of independent sets of the same graph, where the vertices in independent sets are depicted by large black circles (tokens)

Such a sequence is called a *reconfiguration sequence* between  $I_0$  and  $I_r$ . Figure 1 illustrates a reconfiguration sequence  $\langle I_0, I_1, \dots, I_5 \rangle$  of independent sets, which transforms  $I_0$  into  $I_r = I_5$ ; therefore, the answer is “YES” for this instance.

Recently, similar settings of problems have been extensively studied in the framework of reconfiguration problems [9], which arise when we wish to find a step-by-step transformation between two feasible solutions of a problem instance such that all intermediate solutions are also feasible and each step abides by a prescribed reconfiguration rule (i.e., an adjacency relation defined on feasible solutions of the original problem). For example, the TOKEN JUMPING problem can be seen as a reconfiguration problem for the (ordinary) independent set problem: feasible solutions are defined to be all independent sets of the same cardinality in a graph, as in the condition (a) above; and the reconfiguration rule is defined to be the condition (b) above. This reconfiguration framework has been applied to several well-studied combinatorial problems, including independent set [2, 4, 8, 9, 11, 13–16], satisfiability [6], set cover, clique, matching [9], vertex-coloring [1, 3], list edge-coloring [10], (list)  $L(2, 1)$ -labeling [12], and so on.

## 1.1 Known and Related Results

The first reconfiguration problem for independent set, called TOKEN SLIDING, was introduced by Hearn and Demaine [8] which employs another reconfiguration rule. Indeed, there are three reconfiguration problems for independent set, called TOKEN JUMPING [4, 11, 13, 16], TOKEN SLIDING [3, 4, 8, 13, 16], and TOKEN ADDITION AND REMOVAL [2, 9, 13–16]. (See [13] for the definitions.) These are the most intensively studied reconfiguration problems, and hence we here explain only the results strongly related to this paper; see the references above for the other results.

First, TOKEN JUMPING (indeed, all three reconfiguration problems for independent set) is PSPACE-complete for planar graphs of maximum degree three [3, 8, 11], for perfect graphs [13], and for bounded bandwidth graphs [16].

Second, Kamiński et al. [13] gave a linear-time algorithm for TOKEN JUMPING on even-hole-free graphs. Furthermore, their algorithm can find a reconfiguration sequence with the shortest length.

Third, Ito et al. [11] proved that TOKEN JUMPING is  $W[1]$ -hard for general graphs when parameterized only by the number of tokens. Therefore, it is very unlikely that the problem admits a fixed-parameter algorithm for general graphs when the parameter is only the number of tokens. They also gave a fixed-parameter algorithm for general graphs when parameterized by both the number of tokens and the maximum degree of graphs. Their algorithm can be modified so that it finds a reconfiguration sequence with the shortest length.

## 1.2 Our Contribution

In this paper, we first give a fixed-parameter algorithm for TOKEN JUMPING on planar graphs when parameterized only by the number of tokens. Recall that TOKEN JUMPING is PSPACE-complete for planar graphs of maximum degree three, and is  $W[1]$ -hard for general graph when the parameter is only the number of tokens.

Interestingly, our algorithm for planar graphs utilizes only the property that no planar graph contains a complete bipartite graph  $K_{3,3}$  as a subgraph [5]. We show that the same scheme of the algorithm for planar graphs can be applied to a wider class of graphs which forbid a complete bipartite graph  $K_{3,t}$  as a subgraph for a fixed integer  $t \geq 3$ . (We call such graphs  $K_{3,t}$ -forbidden graphs.)

In addition, the algorithm for  $K_{3,t}$ -forbidden graphs (and hence for planar graphs) can be modified so that it finds a reconfiguration sequence with the shortest length for a yes-instance. We note that the reconfiguration sequence in Fig. 1 is shortest. It is remarkable that the token on the vertex  $u$  in Fig. 1(a) must make a “detour” to avoid violating the independence of tokens: it is moved twice even though  $u \in I_0 \cap I_r$ . Our algorithm can capture such detours for  $K_{3,t}$ -forbidden graphs.

## 1.3 Strategy for Fixed-Parameter Algorithms

We here explain two main ideas to develop a fixed-parameter algorithm for TOKEN JUMPING; formal descriptions will be given later.

The first idea is to find a sufficiently large “buffer space” to move the tokens. Namely, we first move all the tokens from  $I_0$  to the buffer space, and then move them from the buffer space to  $I_r$ ; thus, the answer is “YES” if we can find such a buffer space. Due to the usage, such a buffer space (a set of vertices) should be mutually independent and preferably not adjacent to any vertex in  $I_0 \cup I_r$ .

The second idea is to “shrink the graph” into a smaller one with preserving the reconfigurability (i.e., the existence/nonexistence of a reconfiguration sequence) between  $I_0$  and  $I_r$ . This idea is based on the claim that, if the size of the graph is bounded by a function depending only on the parameter  $k$ , we can solve the problem in a brute-force manner in fixed-parameter running time.

Thus, it is useful to find such “removable” vertices in fixed-parameter running time, and shrink the graph so that the size of the resulting graph is bounded by a function of  $k$ .

The  $K_{3,t}$ -forbiddance (and hence  $K_{3,3}$ -forbiddance) of graphs satisfies the two main ideas above at the same time: it ensures that the graph has a sufficiently large independent sets, which may be used as a buffer space; and it characterizes removable vertices.

Due to the page limitation, we omit some proofs from this extended abstract.

## 2 Preliminaries

In this paper, we assume without loss of generality that graphs are simple. Let  $G = (V, E)$  be a graph with vertex set  $V$  and edge set  $E$ . The *order* of  $G$  is the number of vertices in  $G$ . We say that a vertex  $w$  in  $G$  is a *neighbor* of a vertex  $v$  if  $\{v, w\} \in E$ . For a vertex  $v$  in  $G$ , let  $N_G(v) = \{w \in V \mid \{v, w\} \in E\}$ . We also denote  $N_G(v) \cup \{v\}$  by  $N_G[v]$ . For a vertex set  $S \subseteq V$ , let  $N_G(S) = \bigcup_{v \in S} N_G(v)$  and  $N_G[S] = \bigcup_{v \in S} N_G[v]$ .

For a vertex set  $S \subseteq V$  of a graph  $G = (V, E)$ ,  $G[S]$  denotes the subgraph *induced* by  $S$ , that is,  $G[S] = (S, E[S])$  where  $E[S] = \{\{u, v\} \in E \mid \{u, v\} \subseteq S\}$ . A vertex set  $S$  of  $G$  is an *independent set* of  $G$  if  $G[S]$  contains no edge. A subgraph  $G'$  of  $G$  is called a *clique* if every pair of vertices in  $G'$  is joined by an edge; we denote by  $K_s$  a clique of order  $s$ . For two positive integers  $p$  and  $q$ , we denote by  $K_{p,q}$  a complete bipartite graph with its bipartition of size  $p$  and  $q$ .

A graph is *planar* if it can be embedded in the plane without any edge-crossing [5]. In Section 3, our algorithms utilize an independent set of sufficiently large size in a graph, as a buffer space to move tokens. As for independent sets of planar graphs, the following is known, though the original description is about the four-color theorem.

**Proposition 1 ([17]).** *For a planar graph of order  $n = 4s$ , there exists an independent set of size at least  $s$ , and it can be found in  $O(n^2)$  time.*

It is well known as Kuratowski’s theorem that a graph is planar if and only if it does not contain a subdivision of  $K_5$  or  $K_{3,3}$  [5]. Therefore, any planar graph contains neither  $K_5$  nor  $K_{3,3}$  as a subgraph. In this paper, we extend our algorithm for planar graphs to a much larger class of graphs. For two positive integers  $p$  and  $q$ , a graph is  $K_{p,q}$ -*forbidden* if it contains no  $K_{p,q}$  as a subgraph. For example, any planar graph is  $K_{3,3}$ -forbidden. It is important that any  $K_{p,q}$ -forbidden graph contains no clique  $K_{p+q}$  of size  $p + q$ .

In our algorithm for  $K_{3,t}$ -forbidden graphs in Section 3.2, we use Ramsey’s theorem, instead of Proposition 1, to guarantee a sufficiently large independent set. Ramsey’s theorem states that, for every pair of integers  $a$  and  $b$ , there exists an integer  $n$  such that any graph of order at least  $n$  has an independent set of size  $a$  or a clique of size  $b$  (see [7] for example). The smallest number  $n$  of vertices required to achieve this property is called a *Ramsey number*, denoted



by  $\text{Ramsey}(a, b)$ . It is known that  $\text{Ramsey}(a, b) \leq \binom{a+b-2}{b-1}$  [7]. Since any  $K_{p,q}$ -forbidden graph contains no  $K_{p+q}$ , we have the following proposition.

**Proposition 2.** *For integers  $p, q$  and  $s$ , let  $G$  be a  $K_{p,q}$ -forbidden graph of order at least  $\text{Ramsey}(s, p + q)$ . Then,  $G$  has an independent set of size at least  $s$ .*

### 3 Fixed-Parameter Algorithm

In this section, we present a fixed-parameter algorithm for planar graphs to determine if a given TOKEN JUMPING instance is reconfigurable or not, as in the following theorem.

**Theorem 1.** *TOKEN JUMPING with  $k$  tokens can be solved for planar graphs  $G = (V, E)$  in  $O(|E| + (f_1(k))^{2k})$  time, where  $f_1(k) = 2^{6k+1} + 180k^3$ .*

As a proof of Theorem 1, we will prove that Algorithm 1, described below, is such an algorithm. In Section 3.1, we will explain the algorithm step by step, together with its correctness. We will show in Section 3.2 that our algorithm for planar graphs can be extended to that for  $K_{3,t}$ -forbidden graphs,  $t \geq 3$ .

#### 3.1 Planar Graphs

As we have mentioned in Introduction, our algorithm is based on two main ideas: it returns “YES” as soon as we can find a sufficiently large buffer space (Lemmas 1 and 3); otherwise it shrinks the graph so as to preserve the existence/nonexistence of a reconfiguration sequence between two given independent sets  $I_0$  and  $I_r$  (Lemma 4). After shrinking the graph into a smaller one of the order depending only on  $k$ , we can solve the problem in a brute-force manner (Lemma 5). It is important to notice that our algorithm returns “NO” only in this brute-force step. In the following, we explain how the algorithm finds a buffer space or shrinks the graph, which well utilizes the  $K_{3,3}$ -forbiddance of  $G$ .

At the beginning part of the algorithm (lines 1–2), we set two parameters  $\alpha$  and  $\beta$  as  $4k$  and  $10k$ , respectively. These are the orders of (sub)graphs that guarantee the existence of sufficiently large independent sets that will be used as a buffer space. Let  $A = N_G(I_0 \cup I_r) \setminus (I_0 \cup I_r)$ , that is, the set of vertices that are not in  $I_0 \cup I_r$  and have at least one neighbor in  $I_0 \cup I_r$ . Let  $R = V \setminus N_G[I_0 \cup I_r]$ . Then, no vertex in  $R$  is adjacent with any vertex in  $I_0 \cup I_r$ . Notice that  $I_0 \cup I_r$ ,  $A$  and  $R$  form a partition of  $V$ .

**Step 1:** Lines 3–4 of Algorithm 1.

If  $|R| \geq \alpha = 4k$ , then by Proposition 1 the subgraph  $G[R]$  has an independent set of size at least  $k$ . Then, the algorithm returns “YES” because we can use it as a buffer space, as follows.

**Lemma 1.** *If  $|R| \geq \alpha$ , there is a reconfiguration sequence between  $I_0$  and  $I_r$ .*

**Algorithm 1.** TokenJump for planar graphs

**Input:** A parameter  $k$ , a planar graph  $G = (V, E)$ , and two independent sets  $I_0$  and  $I_r$  of  $G$  such that  $|I_0| = |I_r| = k$ .

**Output:** “YES” if there is a reconfiguration sequence between  $I_0$  and  $I_r$ ; otherwise “NO.”

- 1:  $\alpha := 4k, \beta := 10k$ .
- 2:  $A := N_G(I_0 \cup I_r) \setminus (I_0 \cup I_r), R := V \setminus N_G[I_0 \cup I_r]$ .
- 3: **if**  $|R| \geq \alpha$  **then** {Step 1:  $R$  has a sufficiently large buffer space}
- 4:     **return** “YES” and exit.
- 5: **else**  $\{|R| < \alpha$  holds below}
- 6:     **for** each vector  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$  **do**
- 7:          $A(\mathbf{x}) := \{v \in A \mid N_G(v) \cap (V \setminus A) = \text{ON}(\mathbf{x})\}$ .
- 8:         **if**  $|A(\mathbf{x})| \geq \beta$  **then**
- 9:             **if**  $|\text{ON}(\mathbf{x}) \cap I_0| \leq 1$  and  $|\text{ON}(\mathbf{x}) \cap I_r| \leq 1$  **then**
- 10:                 {Step 2:  $A(\mathbf{x})$  has a sufficiently large buffer space}
- 11:                 **return** “YES” and exit.
- 12:             **else** {Step 3: shrink the graph}
- 13:                 Choose an arbitrary subset  $B(\mathbf{x})$  of  $A(\mathbf{x})$  with  $\beta$  vertices, and remove all vertices in  $A(\mathbf{x}) \setminus B(\mathbf{x})$  from  $V$  (and update  $V$ ).
- 14:             **end if**
- 15:         **end if**
- 16:     **end for**{ $|A(\mathbf{x})| \leq \beta$  hold for all vectors  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ }
- 17: **end if**{The order of  $G$  now depends only on  $k$ }
- 18: Check the existence of a reconfiguration sequence in a brute-force manner.

*Proof.* Let  $I'$  be an independent set of  $G[R]$  with  $|I'| \geq k$ ; by Proposition 1 such an independent set  $I'$  always exists. Since no vertex in  $G[R]$  is adjacent with any vertex in  $I_0 \cup I_r$ , there is a reconfiguration sequence between  $I_0$  and  $I_r$  via  $I'$ , as follows: move all tokens on the vertices in  $I_0$  to vertices in  $I'$  one by one; and move all tokens on vertices in  $I'$  to the vertices in  $I_r$  one by one.  $\square$

**Step 2:** Lines 9–11 of Algorithm 1.

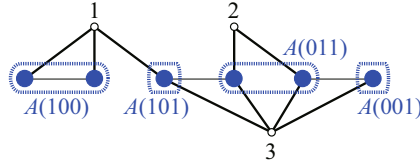
We now know that  $|R| < \alpha$ . Since  $R$  was small, the algorithm then tries to find a sufficiently large buffer space in  $A$ . Notice that

$$|V \setminus A| = |I_0 \cup I_r \cup R| < 2k + \alpha, \tag{1}$$

which depends only on  $k$ . We will partition  $A$  into at most  $2^{2k+\alpha} = 2^{6k}$  subsets, according to how the vertices in  $A$  are adjacent with vertices in  $V \setminus A$ .

Before partitioning  $A$ , we first introduce a new notation. For a vertex set  $S \subseteq V$ , let  $\mathbf{x}$  be an  $|S|$ -dimensional binary vector in  $\{0, 1\}^S$ ; we denote by  $x_v$  the component of  $\mathbf{x}$  corresponding to a vertex  $v \in S$ . For each vector  $\mathbf{x} \in \{0, 1\}^S$ , let  $\text{ON}(\mathbf{x}) = \{v \in S \mid x_v = 1\}$ . For example, if  $\mathbf{x} = (10011) \in \{0, 1\}^{\{1,4,5,6,8\}}$  for a vertex set  $S = \{1, 4, 5, 6, 8\}$ , then  $x_1 = 1, x_4 = 0, x_5 = 0, x_6 = 1, x_8 = 1$  and  $\text{ON}(\mathbf{x}) = \{1, 6, 8\}$ .

To partition the vertex set  $A$ , we prepare all binary vectors in  $\{0, 1\}^{V \setminus A}$ . By Eq. (1) the number of the prepared vectors is at most  $2^{2k+\alpha} = 2^{6k}$ . For each



**Fig. 2.** Partitioning  $A$  of six large (blue) vertices into four subsets  $A(100)$ ,  $A(101)$ ,  $A(011)$  and  $A(001)$ , where each vertex in  $V \setminus A = \{1, 2, 3\}$  is represented by a small (white) circle

vector  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ , we define  $A(\mathbf{x}) = \{v \in A \mid N_G(v) \cap (V \setminus A) = \text{ON}(\mathbf{x})\}$ , that is,  $\mathbf{x}$  is used to represent a pattern of neighbors in  $V \setminus A$ . (See Fig. 2.) Therefore, all vertices in the same subset  $A(\mathbf{x})$  have exactly the same neighbors in  $V \setminus A = I_0 \cup I_r \cup R$ , that is, the vertices in  $\text{ON}(\mathbf{x})$ . Conversely, each vertex in  $\text{ON}(\mathbf{x})$  is adjacent with *all* vertices in  $A(\mathbf{x})$ . We thus have the following proposition.

**Proposition 3.** *For each vector  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ ,  $G[A(\mathbf{x}) \cup \text{ON}(\mathbf{x})]$  contains a complete bipartite graph  $K_{|A(\mathbf{x})|, |\text{ON}(\mathbf{x})|}$  as a subgraph whose bipartition consists of  $A(\mathbf{x})$  and  $\text{ON}(\mathbf{x})$ .*

Note that 0-vector (i.e., every component is 0) is not used, because each vertex in  $A$  is adjacent to at least one vertex in  $I_0 \cup I_r$ . In this way, we partition  $A$  into at most  $2^{2k+\alpha}$  subsets  $A(\mathbf{x})$  according to the vectors  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ . Proposition 3 and the  $K_{3,3}$ -forbiddance give the following property on  $\text{ON}(\mathbf{x})$ . (Note that  $\beta \geq 3$ .)

**Lemma 2.** *If  $|A(\mathbf{x})| \geq \beta$  holds for a vector  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ , then  $|\text{ON}(\mathbf{x})| \leq 2$ .*

The algorithm tries to find a sufficiently large buffer space from one of the subsets  $A(\mathbf{x})$ ,  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ , such that  $|A(\mathbf{x})| \geq \beta = 10k$ . The following lemma proves the correctness of Step 2.

**Lemma 3.** *Suppose that there exists a binary vector  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$  such that  $|A(\mathbf{x})| \geq \beta$ ,  $|\text{ON}(\mathbf{x}) \cap I_0| \leq 1$  and  $|\text{ON}(\mathbf{x}) \cap I_r| \leq 1$ . Then, there exists a reconfiguration sequence between  $I_0$  and  $I_r$ .*

*Proof.* Suppose that such a vector  $\mathbf{x}$  exists. Since  $|A(\mathbf{x})| \geq \beta = 10k$ , by Proposition 1 the graph  $G[A(\mathbf{x})]$  has an independent set  $I'$  of size at least  $2k$ . Let  $w_0 \in \text{ON}(\mathbf{x}) \cap I_0$  and  $w_r \in \text{ON}(\mathbf{x}) \cap I_r$  if such vertices exist;  $w_0 = w_r$  may hold. Then, we obtain a reconfiguration sequence between  $I_0$  and  $I_r$  via  $I'$ , as follows:

- (a) move the token on  $w_0$  to an arbitrary vertex  $w'$  in  $I'$ ;
- (b) move all tokens in  $I_0 \setminus \{w_0\}$  to vertices in  $I' \setminus \{w'\}$  one by one;
- (c) move tokens in  $I' \setminus \{w'\}$  to the vertices in  $I_r \setminus \{w_r\}$  one by one; and
- (d) move the last token on  $w' \in I'$  to  $w_r$ .

Note that no vertex in  $I_0 \setminus \{w_0\}$  is adjacent with any vertex in  $I'$  because  $|\text{ON}(\mathbf{x}) \cap I_0| \leq 1$ . Furthermore, since  $I'$  is an independent set in  $G[A(\mathbf{x})]$ ,  $w' \in I'$  is not adjacent with any vertex in  $I' \setminus \{w'\}$ . Therefore, we can execute both (a) and (b) above without violating the independence of tokens. By the symmetric

arguments, we can execute both (c) and (d) above, too. Thus, according to (a)–(d) above, there exists a reconfiguration sequence between  $I_0$  and  $I_r$ .  $\square$

**Step 3:** Line 13 of Algorithm 1.

We now consider to shrink the graph: the algorithm shrinks each subset  $A(\mathbf{x})$  of size more than  $\beta$  into a smaller one  $B(\mathbf{x})$  of size  $\beta$ .

Consider any subset  $A(\mathbf{x})$  of size more than  $\beta$ . Then, by Lemma 2 we have  $|\text{ON}(\mathbf{x})| \leq 2$ . In fact, since we have executed Step 2,  $|\text{ON}(\mathbf{x}) \cap I_0| = 2$  or  $|\text{ON}(\mathbf{x}) \cap I_r| = 2$  holds (recall Lemma 3). We choose an arbitrary set  $B(\mathbf{x})$  of  $\beta = 10k$  vertices from  $A(\mathbf{x})$ . Then, localizing independent sets intersecting  $A(\mathbf{x})$  only to  $B(\mathbf{x})$  does not affect the reconfigurability, as in the following lemma.

**Lemma 4.**  *$G$  has a reconfiguration sequence between  $I_0$  and  $I_r$  if and only if there exists a reconfiguration sequence  $\langle I_0, I'_1, I'_2, \dots, I'_{\ell'}, I_r \rangle$  such that  $I'_j \cap A(\mathbf{x}) \subseteq B(\mathbf{x})$  holds for every index  $j \in \{1, 2, \dots, \ell'\}$ .*

*Proof.* The if-part clearly holds, and hence we prove the only-if-part. Suppose that  $G$  has a reconfiguration sequence  $\langle I_0, I_1, \dots, I_{\ell}, I_r \rangle$  between  $I_0$  and  $I_r$ . If  $I_j \cap A(\mathbf{x}) \subseteq B(\mathbf{x})$  holds for every  $j \in \{1, 2, \dots, \ell\}$ , the claim is already satisfied. Thus, let  $I_p$  and  $I_q$  be the first and last independent sets of  $G$ , respectively, in the subsequence  $\langle I_1, I_2, \dots, I_{\ell} \rangle$  such that  $I_j \cap (A(\mathbf{x}) \setminus B(\mathbf{x})) \neq \emptyset$ . Note that  $p = q$  may hold. Then,  $I_p$  contains exactly one vertex  $w_p$  in  $I_p \cap (A(\mathbf{x}) \setminus B(\mathbf{x}))$ , and  $I_q$  contains exactly one vertex  $w_q$  in  $I_q \cap (A(\mathbf{x}) \setminus B(\mathbf{x}))$ ;  $w_p = w_q$  may hold. It should be noted that both  $\text{ON}(\mathbf{x}) \cap I_p = \emptyset$  and  $\text{ON}(\mathbf{x}) \cap I_q = \emptyset$  hold, because  $w_p, w_q \in A(\mathbf{x})$  are both adjacent with the two vertices in  $\text{ON}(\mathbf{x})$ .

We first claim that  $G[B(\mathbf{x})]$  contains an independent set  $I^*$  such that  $|I^*| \geq k$  and  $I^* \cap N_G[I_p \cup I_q] = \emptyset$ . By Proposition 3,  $G[A(\mathbf{x}) \cup \text{ON}(\mathbf{x})]$  contains a complete bipartite graph  $K_{|A(\mathbf{x})|, 2}$  as a subgraph; recall that  $|\text{ON}(\mathbf{x})| = 2$ . Therefore, due to the  $K_{3,3}$ -forbiddance of  $G$ , every vertex in  $V \setminus \text{ON}(\mathbf{x})$  can be adjacent with at most two vertices in  $A(\mathbf{x})$ , and hence at most two vertices in  $B(\mathbf{x})$ . Since both  $\text{ON}(\mathbf{x}) \cap I_p = \emptyset$  and  $\text{ON}(\mathbf{x}) \cap I_q = \emptyset$  hold, we have  $I_p, I_q \subseteq V \setminus \text{ON}(\mathbf{x})$  and hence

$$|B(\mathbf{x}) \setminus N_G[I_p \cup I_q]| \geq \beta - 3 \cdot |I_p \cup I_q| \geq 10k - 6k = 4k. \quad (2)$$

By Proposition 1 we conclude that  $G[B(\mathbf{x}) \setminus N_G[I_p \cup I_q]]$  contains an independent set  $I^*$  such that  $|I^*| \geq k$  and  $I^* \cap N_G[I_p \cup I_q] = \emptyset$ .

We then show that the subsequence  $\langle I_p, I_{p+1}, \dots, I_q \rangle$  can be replaced with another sequence  $\langle \tilde{I}_p, \tilde{I}_{p+1}, \dots, \tilde{I}_{q'} \rangle$  such that  $\tilde{I}_j \cap A(\mathbf{x}) \subseteq B(\mathbf{x})$  for every  $j \in \{p, p+1, \dots, q'\}$ . To do so, we use the independent set  $I^*$  of  $G[B(\mathbf{x}) \setminus N_G[I_p \cup I_q]]$  as a buffer space. We now explain how to move the tokens:

- (1) From the independent set  $I_{p-1}$ , we first move the token on the vertex  $u_p$  in  $I_{p-1} \setminus I_p$  to an arbitrarily chosen vertex  $v^*$  in  $I^*$ , instead of  $w_p$ . Let  $\tilde{I}_p$  be the resulting vertex set, that is,  $\tilde{I}_p = (I_p \setminus \{w_p\}) \cup \{v^*\}$ .
- (2) We then move all tokens in  $I_{p-1} \setminus \{u_p\}$  ( $= I_{p-1} \cap I_p$ ) to vertices in  $I^* \setminus \{v^*\}$  one by one in an arbitrary order.

- (3) We move tokens in  $I^* \setminus \{v^*\}$  to the vertices in  $I_q \cap I_{q+1}$  ( $= I_q \setminus \{w_q\}$ ) one by one in an arbitrary order. Let  $\tilde{I}_{q'}$  be the resulting vertex set, that is,  $\tilde{I}_{q'} = (I_q \setminus \{w_q\}) \cup \{v^*\} = (I_q \cap I_{q+1}) \cup \{v^*\}$ .

Clearly,  $\tilde{I}_j \cap A(\mathbf{x}) \subseteq B(\mathbf{x})$  holds for every  $j \in \{p, \dots, q'\}$ . Furthermore, notice that  $I_{q+1}$  can be obtained from  $\tilde{I}_{q'}$  by moving a single token on  $v^* \in \tilde{I}_{q'}$  to the vertex in  $I_{q+1} \setminus I_q$ . Therefore,  $\langle I_p, I_{p+1}, \dots, I_q \rangle$  can be correctly replaced with  $\langle \tilde{I}_p, \tilde{I}_{p+1}, \dots, \tilde{I}_{q'} \rangle$  if the vertex set  $\tilde{I}_j$  forms an independent set of  $G$  for every  $j \in \{p, \dots, q'\}$ . For every  $j \in \{p, \dots, q'\}$ , notice that either  $\tilde{I}_j \subset I_p \cup I^*$  or  $\tilde{I}_j \subset I_q \cup I^*$  holds. Then, since  $I^* \cap N_G[I_p \cup I_q] = \emptyset$ , the vertex set  $\tilde{I}_j$  forms an independent set of  $G$ .

By this way, we obtain a reconfiguration sequence  $\langle I_0, \dots, I_{p-1}, \tilde{I}_p, \dots, \tilde{I}_{q'}, I_{q+1}, \dots, I_r \rangle$  such that no vertex in  $A(\mathbf{x}) \setminus B(\mathbf{x})$  is contained in any independent set in the sequence.  $\square$

Lemma 4 implies that, even if we remove all vertices in  $A(\mathbf{x}) \setminus B(\mathbf{x})$  for an arbitrary chosen set  $B(\mathbf{x}) \subseteq A(\mathbf{x})$  of  $\beta = 10k$  vertices, it does not affect the existence/nonexistence of a reconfiguration sequence between  $I_0$  and  $I_r$ . Thus, we can shrink the subset  $A(\mathbf{x})$  into  $B(\mathbf{x})$  of size  $\beta = 10k$ .

**Step 4:** Line 18 of Algorithm 1.

In this step,  $|A(\mathbf{x})| \leq \beta = 10k$  hold for all vectors  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ . Furthermore, Proposition 3 and the  $K_{3,3}$ -forbiddance of  $G$  imply that  $|A(\mathbf{x})| \leq 2$  if  $|\text{ON}(\mathbf{x})| \geq 3$ . Since  $\alpha = 4k$  and  $\beta = 10k$ , by Eq. (1) we have

$$\begin{aligned} |A| &= \sum \{ |A(\mathbf{x})| : \mathbf{x} \in \{0, 1\}^{V \setminus A}, 1 \leq |\text{ON}(\mathbf{x})| \leq 2 \} \\ &\quad + \sum \{ |A(\mathbf{x})| : \mathbf{x} \in \{0, 1\}^{V \setminus A}, |\text{ON}(\mathbf{x})| \geq 3 \} \\ &\leq \beta \cdot \left( (2k + \alpha) + \binom{2k + \alpha}{2} \right) + 2 \cdot \left( 2^{2k + \alpha} - (2k + \alpha) - \binom{2k + \alpha}{2} \right) \\ &= 2^{6k + 1} + 180k^3 - 6k^2 - 6k. \end{aligned}$$

Then, since  $|I_0 \cup I_r| \leq 2k$  and  $|R| \leq \alpha = 4k$ , we can bound  $|V|$  by

$$|V| = |I_0 \cup I_r| + |R| + |A| < 2^{6k + 1} + 180k^3,$$

which is denoted by  $f_1(k)$ . Since the order  $f_1(k)$  of  $G$  now depends only on  $k$ , we can apply a brute-force algorithm as follows.

**Lemma 5.** *If  $|V| \leq f_1(k)$ , TOKEN JUMPING is solvable in  $O\left((f_1(k))^{2k}\right)$  time.*

*Proof.* We construct a configuration graph  $\mathcal{C} = (\mathcal{V}, \mathcal{E})$ , as follows:

- (i) each node in  $\mathcal{C}$  corresponds to an independent set of  $G$  with size  $k$ ; and
- (ii) two nodes in  $\mathcal{C}$  are joined by an edge if and only if the corresponding two independent sets can be reconfigured by just a single token jump.

Clearly, there is a reconfiguration sequence between  $I_0$  and  $I_r$  if and only if there is a path in  $\mathcal{C}$  between the two corresponding nodes.

Since  $G$  has at most the number  $\binom{f_1(k)}{k}$  of distinct independent sets of size exactly  $k$ , we have  $|\mathcal{V}| \leq (f_1(k))^k$ . The configuration graph  $\mathcal{C}$  above can be constructed in  $O(|\mathcal{V}|^2)$  time. Furthermore, by the breadth-first search on  $\mathcal{C}$  which starts from the node corresponding to  $I_0$ , we can check if  $\mathcal{C}$  has a desired path or not in  $O(|\mathcal{V}| + |\mathcal{E}|) = O(|\mathcal{V}|^2)$  time. In this way, TOKEN JUMPING can be solved in  $O(|\mathcal{V}|^2) = O((f_1(k))^{2k})$  time in total.  $\square$

This completes the correctness proof of Algorithm 1.

**Running Time**

We now estimate the running time of Algorithm 1. We first claim that lines 1–17 can be executed in  $O(|E|)$  time. Lines 6–16 can be clearly done in fixed-parameter running time, but actually these lines can be done in  $O(|E|)$  time because  $|A| = |\bigcup\{A(\mathbf{x}) : \mathbf{x} \in \{0, 1\}^{V \setminus A}\}|$  is at most  $n$ ; we can compute  $\mathbf{x}$  implicitly. By Lemma 5 we can execute line 18 in  $O((f_1(k))^{2k})$  time. Thus, the total running time of Algorithm 1 is  $O(|E| + (f_1(k))^{2k})$ .

This completes the proof of Theorem 1.

**3.2  $K_{3,t}$ -Forbidden Graphs**

In this subsection, we show that our algorithm for planar graphs can be extended to that for  $K_{3,t}$ -forbidden graphs, and give the following theorem.

**Theorem 2.** *For a fixed integer  $t \geq 3$ , let  $G$  be a  $K_{3,t}$ -forbidden graph. Then, TOKEN JUMPING for  $G$  can be solved in fixed-parameter running time, when parameterized by the number  $k$  of tokens.*

We here give a sketch of how to adapt the fixed-parameter algorithm for planar graphs in Section 3.1 to  $K_{3,t}$ -forbidden graphs.

The first point is to set two parameters  $\alpha_t$  and  $\beta_t$  that correspond to  $\alpha$  and  $\beta$ , respectively. Recall that  $\alpha = 4k$  and  $\beta = 10k$  are the orders of (sub)graphs that guarantee the existence of sufficiently large independent sets that will be used as a buffer space. For  $K_{3,t}$ -forbidden graphs, we set  $\alpha_t = \text{Ramsey}(k, t + 3)$  and  $\beta_t = \text{Ramsey}((2t + 1)k, t + 3)$ . Then, Proposition 2 guarantees the existence of independent sets of size  $k$  in  $R$ , and hence Step 1 of Algorithm 1 can be adapted to  $K_{3,t}$ -forbidden graphs. We note that, although no exact formula of Ramsey number is known, we can bound it from above, say  $\text{Ramsey}(a, b) \leq \binom{a+b-2}{b-1}$  [7]. Therefore, we indeed set  $\alpha_t = (k + t + 1)^{t+2}$  and  $\beta_t = ((2t + 1)k + t + 1)^{t+2}$ , both of which are fixed-parameter size.

The second point is to extend Lemma 2 for planar graphs to that for  $K_{3,t}$ -forbidden graphs, as follows. (Note that  $\beta_t \geq t$ .)

**Lemma 6.** *If  $|A(\mathbf{x})| \geq \beta_t$  holds for a vector  $\mathbf{x} \in \{0, 1\}^{V \setminus A}$ , then  $|\text{ON}(\mathbf{x})| \leq 2$ .*

Then, since there is an independent set of size at least  $(2t + 1)k$  in  $A(\mathbf{x})$ , Step 2 of Algorithm 1 can be adapted to  $K_{3,t}$ -forbidden graphs.

The third point is to modify Eq. (2) in the proof of Lemma 4 and shrink  $A(\mathbf{x})$  to size  $\beta_t$ . Recall that the vertices in  $\text{ON}(\mathbf{x})$  and  $A(\mathbf{x})$  form a complete bipartite graph  $K_{2,|A(\mathbf{x})|}$ , and hence any vertex other than  $\text{ON}(\mathbf{x})$  can be adjacent with at most  $(t-1)$  vertices in  $A(\mathbf{x})$ , due to the  $K_{3,t}$ -forbiddance of the graph. Therefore, if  $A(\mathbf{x})$  has an independent set of size at least  $(2t+1)k$ , we still have at least  $k$  vertices that can be used as a buffer space. Therefore, Lemma 4 can be adapted to  $K_{3,t}$ -forbidden graphs, and hence Step 3 of Algorithm 1 can be, too.

The running time of the adapted algorithm depends on the order of the graph shrunk by Step 3. By the similar arguments for planar graphs, the order of the shrunk graph depends only on  $\alpha_t$  and  $\beta_t$ . Since both  $\alpha_t$  and  $\beta_t$  are fixed-parameter size, the adapted algorithm runs in fixed-parameter running time.

This completes the proof of Theorem 2.

## 4 Shortest Reconfiguration Sequence

In the previous section, we present an algorithm which simply determines if there exists a reconfiguration sequence between two given independent sets  $I_0$  and  $I_r$ . If the answer is yes, it is natural to consider how we actually move tokens on  $I_0$  to  $I_r$ . For this question, it is easy to modify Algorithm 1 to output an actual reconfiguration sequence, although it is not always shortest. In this section, we consider how to move tokens on  $I_0$  to  $I_r$  in a shortest way.

**Theorem 3.** *For a fixed integer  $t \geq 3$ , let  $G$  be a  $K_{3,t}$ -forbidden graph. Given a yes-instance of TOKEN JUMPING on  $G$ , a shortest reconfiguration sequence can be found in fixed-parameter running time, where the parameter is the number  $k$  of tokens.*

*Proof sketch.* We explain how to modify Algorithm 1 so as to find a shortest reconfiguration sequence. The biggest change from Algorithm 1 is that the modified algorithm does not stop until Step 4. Algorithm 1 can exit at Steps 1 and 2 after finding a buffer space, which means that there exists a reconfiguration sequence from  $I_0$  to  $I_r$  via vertices only in  $R$  and vertices only in  $A(\mathbf{x})$ , respectively. However, this does not directly imply the existence of a *shortest* reconfiguration sequence from  $I_0$  to  $I_r$  that uses vertices only in  $R$  (or only in  $A(\mathbf{x})$ ). Thus, we do not exit at Steps 1 and 2, but shrink  $R$  and  $A(\mathbf{x})$  of the original graph into a fixed-parameter size so as to preserve the shortest length of a reconfiguration sequence in the original graph; then we can find a shortest reconfiguration sequence in Step 4 by the brute-force algorithm proposed in Lemma 5. (Details are omitted from this extended abstract.)  $\square$

**Acknowledgments.** We are grateful to Akira Suzuki, Ryuhei Uehara, and Katsuhisa Yamanaka for fruitful discussions with them. We thank anonymous referees for their helpful suggestions. This work is partially supported by JSPS KAKENHI 25106504 and 25330003 (T. Ito), and 25104521, 26540005 and 26540005 (H. Ono).

## References

1. Bonamy, M., Johnson, M., Lignos, I., Patel, V., Paulusma, D.: Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *J. Combinatorial Optimization* **27**, 132–143 (2014)
2. Bonsma, P.: Independent set reconfiguration in cographs. In: WG 2014 (to appear, 2014). [arXiv:1402.1587](https://arxiv.org/abs/1402.1587)
3. Bonsma, P., Cereceda, L.: Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science* **410**, 5215–5226 (2009)
4. Bonsma, P., Kamiński, M., Wrochna, M.: Reconfiguring Independent Sets in Claw-Free Graphs. In: Ravi, R., Gørtz, I.L. (eds.) SWAT 2014. LNCS, vol. 8503, pp. 86–97. Springer, Heidelberg (2014)
5. Brandstädt, A., Le, V.B., Spinrad, J.P.: *Graph Classes: A Survey*. SIAM (1999)
6. Gopalan, P., Kolaitis, P.G., Maneva, E.N., Papadimitriou, C.H.: The connectivity of Boolean satisfiability: Computational and structural dichotomies. *SIAM J. Computing* **38**, 2330–2355 (2009)
7. Graham, R.L., Rothschild, B.L.: *Ramsey Theory*, 2nd edn. Wiley-Interscience, New York (1990)
8. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science* **343**, 72–96 (2005)
9. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C.H., Sideri, M., Uehara, R., Uno, Y.: On the complexity of reconfiguration problems. *Theoretical Computer Science* **412**, 1054–1065 (2011)
10. Ito, T., Kamiński, M., Demaine, E.D.: Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics* **160**, 2199–2207 (2012)
11. Ito, T., Kamiński, M., Ono, H., Suzuki, A., Uehara, R., Yamanaka, K.: On the Parameterized Complexity for Token Jumping on Graphs. In: Gopal, T.V., Agrawal, M., Li, A., Cooper, S.B. (eds.) TAMC 2014. LNCS, vol. 8402, pp. 341–351. Springer, Heidelberg (2014)
12. Ito, T., Kawamura, K., Ono, H., Zhou, X.: Reconfiguration of list  $L(2, 1)$ -labelings in a graph. *Theoretical Computer Science* **544**, 84–97 (2014)
13. Kamiński, M., Medvedev, P., Milanič, M.: Complexity of independent set reconfigurability problems. *Theoretical Computer Science* **439**, 9–15 (2012)
14. Mouawad, A.E., Nishimura, N., Raman, V., Simjour, N., Suzuki, A.: On the Parameterized Complexity of Reconfiguration Problems. In: Gutin, G., Szeider, S. (eds.) IPEC 2013. LNCS, vol. 8246, pp. 281–294. Springer, Heidelberg (2013)
15. Mouawad, A.E., Nishimura, N., Raman, V., Wrochna, M.: Reconfiguration over tree decompositions. [arXiv:1405.2447](https://arxiv.org/abs/1405.2447)
16. Wrochna, M.: Reconfiguration in bounded bandwidth and treedepth. [arXiv:1405.0847](https://arxiv.org/abs/1405.0847) (2014)
17. Robertson, N., Sanders, D.P., Seymour, P., Thomas, R.: Efficiently four-coloring planar graphs. *Proc. of STOC* **1996**, 571–575 (1996)



# Covering Problems for Partial Words and for Indeterminate Strings

Maxime Crochemore<sup>1,2</sup>, Costas S. Iliopoulos<sup>1,3</sup>, Tomasz Kociumaka<sup>4(✉)</sup>,  
Jakub Radoszewski<sup>4</sup>, Wojciech Rytter<sup>4,5</sup>, and Tomasz Waleń<sup>4</sup>

<sup>1</sup> Department of Informatics, King's College London, London, UK  
{maxime.crochemore,c.iliopoulos}@kcl.ac.uk

<sup>2</sup> Université Paris-Est, Marne-la-Vallée, France

<sup>3</sup> Faculty of Engineering, Computing and Mathematics, University of Western  
Australia, Perth, Australia

<sup>4</sup> Faculty of Mathematics, Informatics and Mechanics, University of Warsaw,  
Warsaw, Poland

{kociumaka,jrad,rytter,walen}@mimuw.edu.pl

<sup>5</sup> Faculty of Mathematics and Computer Science, Copernicus University,  
Toruń, Poland

**Abstract.** We consider the problem of computing a solid cover of an indeterminate string. An indeterminate string may contain non-solid symbols, each of which specifies a subset of the alphabet that could be present at the corresponding position. We also consider covering partial words, which are a special case of indeterminate strings where each non-solid symbol is a don't care symbol. We prove that both indeterminate string covering problem and partial word covering problem are NP-complete for binary alphabet and show that both problems are fixed-parameter tractable with respect to  $k$ , the number of non-solid symbols. For the indeterminate string covering problem we obtain a  $2^{\mathcal{O}(k \log k)} + nk^{\mathcal{O}(1)}$ -time algorithm. For the partial word covering problem we obtain a  $2^{\mathcal{O}(\sqrt{k} \log k)} + nk^{\mathcal{O}(1)}$ -time algorithm. We prove that, unless the Exponential Time Hypothesis is false, no  $2^{\mathcal{O}(\sqrt{k})} n^{\mathcal{O}(1)}$ -time solution exists for this problem, which shows that our algorithm for this case is close to optimal. We also present an algorithm for both problems which is feasible in practice.

## 1 Introduction

A classic string is a sequence of symbols from a given alphabet  $\Sigma$ . In an *indeterminate* string, some positions may contain, instead of a single symbol from  $\Sigma$  (called a *solid* symbol), a subset of  $\Sigma$ . Such a *non-solid* symbol indicates that the exact symbol at the given position is not known, but is suspected to be one

---

Tomasz Kociumaka: Supported by Polish budget funds for science in 2013-2017 as a research project under the 'Diamond Grant' program.

Jakub Radoszewski: The author receives financial support of Foundation for Polish Science.



Throughout the paper we use the following notations:  $n$  for the length of the given indeterminate string,  $k$  for the number of non-solid symbols in the input, and  $\sigma$  for the size of the alphabet. We assume that  $2 \leq \sigma \leq n$  and that each non-solid symbol in the indeterminate string is represented by a bit vector of size  $\sigma$ . Thus the size of the input is  $\mathcal{O}(n + \sigma k)$ .

The first attempts to the problem of indeterminate string covering were made in [2, 5, 11]. However, they considered indeterminate strings as covers and presented some partial results for this case. The common assumption of these papers is that  $\sigma = \mathcal{O}(1)$ ; moreover, in [2, 5] the authors considered only so-called conservative indeterminate strings, for which  $k = \mathcal{O}(1)$ .

**Our Results:** In Section 3 we show an  $\mathcal{O}(n\sigma^{k/2}k^{\mathcal{O}(1)})$ -time algorithm for covering indeterminate strings with a simple implementation. In Section 4 we obtain an  $\mathcal{O}(2^{\mathcal{O}(k \log k)} + nk^{\mathcal{O}(1)})$ -time algorithm. In the same section we devise a more efficient solution for partial words with  $\mathcal{O}(2^{\mathcal{O}(\sqrt{k} \log k)} + nk^{\mathcal{O}(1)})$ -time complexity. Finally in Section 5 we show that both problems are NP-complete already for binary alphabet. As a by-product we obtain that under the Exponential Time Hypothesis no  $\mathcal{O}(2^{o(\sqrt{k})}n^{\mathcal{O}(1)})$ -time solution exists for both problems.

## 2 Preliminaries

An *indeterminate string* (*i-string*, for short)  $T$  of length  $|T| = n$  over a finite alphabet  $\Sigma$  is a sequence  $T[1] \dots T[n]$  such that  $T[i] \subseteq \Sigma$ ,  $T[i] \neq \emptyset$ . If  $|T[i]| = 1$ , that is,  $T[i]$  represents a single symbol of  $\Sigma$ , we say that  $T[i]$  is a *solid* symbol. For convenience we often write that  $T[i] = c$  instead of  $T[i] = \{c\}$  in this case ( $c \in \Sigma$ ). Otherwise we say that  $T[i]$  is a *non-solid* symbol. In what follows by  $k$  we denote the number of non-solid symbols in the considered i-string  $T$  and by  $\sigma$  we denote  $|\Sigma|$ . If  $k = 0$ , we call  $T$  a (solid) string. We say that two i-strings  $U$  and  $V$  *match* (denoted as  $U \approx V$ ) if  $|U| = |V|$  and for each  $i = 1, \dots, |U|$  we have  $U[i] \cap V[i] \neq \emptyset$ .

*Example 1.* Let  $A = a\{b, c\}$ ,  $B = a\{a, b\}$ ,  $C = aa$  be indeterminate strings ( $C$  is a solid string). Then  $A \approx B$ ,  $B \approx C$ , however,  $A \not\approx C$ .

If all  $T[i]$  are either solid or equal to  $\Sigma$  then  $T$  is called a *partial word*. In this case the non-solid “don’t care” symbol is denoted as  $\diamond$ .

By  $T[i..j]$  we denote a factor  $T[i] \dots T[j]$  of  $T$ . If  $i = 1$  then it is called a prefix and if  $j = n$  then it is called a suffix. We say that a pattern i-string  $S$  occurs in a text i-string  $T$  at position  $j$  if  $S$  matches  $T[j..j + |S| - 1]$ . A *border* of  $T$  is a solid string which matches both a prefix and a suffix of  $T$ . A border-length of  $T$  is a positive integer equal to the length of a border of  $T$ . A *cover* of  $T$  is a solid string  $S$  such that, for each  $i = 1, \dots, n$ , there exists an occurrence of  $S$  in  $T$  that contains the position  $i$ , i.e., an occurrence of  $S$  at one of the positions  $\{i - |S| + 1, \dots, i\}$ . Note that, just as in solid strings, every cover of  $T$  is also a border of  $T$ .

*Example 2.* The shortest cover of an i-string  $T$  need not be one of the shortest covers of the solid strings matching  $T$ . E.g., for the i-string  $a\blacklozenge b$ , where  $\blacklozenge = \{a, b\}$ , the shortest cover  $ab$  has length 2, whereas none of the solid strings  $aab$ ,  $abb$  has a cover of length 2.

The following simple observation is an important tool in our algorithms.

**Observation 3.** *There are at most  $k^2$  values (shifts)  $i \in \{1, \dots, n\}$  such that  $T[1 + \ell]$  and  $T[i + \ell]$  are both non-solid for some  $\ell$ .*

For convenience, we compute the set  $T[i] \cap T[j]$  for each pair  $T[i], T[j]$  of non-solid symbols of  $T$ , and label different sets with different integers, so that afterwards we can refer to any of them in  $\mathcal{O}(1)$  space. In particular, after such  $\mathcal{O}(\sigma k^2)$ -time preprocessing, we can check in  $\mathcal{O}(1)$  time if any two positions of  $T$  match.

A longest common prefix (LCP) query in  $T$ , denoted as  $\text{lcp}(i, j)$ , is a query for the longest matching prefix of the i-strings  $T[i..n]$  and  $T[j..n]$ . Recall that for a solid string we can construct in  $\mathcal{O}(n)$  time a data structure that answers LCP-queries in  $\mathcal{O}(1)$  time, see [8]. Note that an LCP-query in an i-string can be reduced to  $\mathcal{O}(k)$  LCP-queries in a solid string:

**Lemma 4.** *For an i-string with  $k$  non-solid symbols, after  $\mathcal{O}(nk^2)$ -time preprocessing, one can compute the LCP of any two positions in  $\mathcal{O}(k)$  time.*

Lemma 4 lets us efficiently check if given pairs of factors of an i-string match and thus it has useful consequences.

**Corollary 5.** *Given i-strings  $S$  and  $T$  of total length  $n$  containing  $k$  non-solid symbols in total, one can compute  $\text{Occ}(S, T)$ , the list of all positions where  $S$  occurs in  $T$ , in  $\mathcal{O}(nk^2)$  time.*

**Corollary 6.** *The set of all border-lengths of an i-string can be computed in  $\mathcal{O}(nk^2)$  time.*

Note that a solid string of length at least  $\frac{n}{2}$  is a cover of  $T$  if and only if it is a border of  $T$ . Therefore Corollary 6 enables us to easily solve the covering problem for cover lengths at least half of the word length. In the following sections we search only for the covers of length at most  $\lfloor \frac{n}{2} \rfloor$ .

### 3 Algorithm Parameterized by $k$ and $\sigma$

Let  $T$  be an i-string of length  $n$  with  $k$  non-solid symbols. We assume that  $T[1.. \lfloor \frac{n}{2} \rfloor]$  contains at most  $k/2$  holes (otherwise we reverse the i-string).

We say that  $S$  is a *solid prefix* of  $T$  if  $S$  is a solid string that matches  $T[1..|S|]$ . For an increasing list of integers  $L = [i_1, i_2, i_3, \dots, i_m]$ ,  $m \geq 2$ , we define

$$\text{maxgap}(L) = \max\{i_{t+1} - i_t : t = 1, \dots, m - 1\}.$$

A set  $\mathcal{P} \subseteq \text{Occ}(S, T)$  is a *covering set* for  $S$  if  $\text{maxgap}(\mathcal{P} \cup \{n+1\}) \leq |S|$ , i.e., the occurrences of  $S$  at positions in  $\mathcal{P}$  already cover the whole text  $T$ .

We introduce a *ShortestCover*( $S, L$ ) subroutine which, for a given solid prefix  $S$  of  $T$  and an increasing list of positions  $L$ , checks if there is a cover of  $T$  which is a prefix of  $S$  for which the covering set is a sublist of  $L$  and, if so, returns the length of the shortest such cover. A pseudocode of this operation can be found on the next page. Correctness of the algorithm follows from the fact that

$$\text{ShortestCover}(S, L) = \min \left\{ j : \text{maxgap} \left( \bigcup_{t \geq j} L_t \cup \{n+1\} \right) \leq j \right\}.$$

**Algorithm** *ShortestCover*( $S, L$ )

**Input:**  $S$ : a solid prefix of  $T$ ;  $L$ : a sublist of  $\{1, \dots, n\}$

**Output:** The length of the shortest cover which is a prefix of  $S$  and has a covering set being a sublist of  $L$

**preprocessing:**

**foreach**  $i \in L$  **do**  $\text{dist}[i] := \text{lcp}(S, T[i..n]);$

$D := \{ \text{dist}[i] : i \in L \};$

**foreach**  $j \in D$  **do**  $L_j := \{ i \in L : \text{dist}[i] = j \};$

$L := L \cup \{n+1\};$

**processing:**

**foreach**  $j \in D$  *in increasing order* **do**

**if**  $\text{maxgap}(L) \leq j$  **then return**  $\text{maxgap}(L);$

**foreach**  $i \in L_j$  **do** remove  $i$  from  $L;$

**return** *no solution*;

**Lemma 7.** *The algorithm ShortestCover*( $S, L$ ) *works in*  $\mathcal{O}(nk)$  *time assuming that the data structure of Lemma 4 is accessible.*

*Proof.* Assume that each time we remove an element from the list we update  $\text{maxgap}(L)$ . Then  $\text{maxgap}(L)$  may only increase. Each operation on the list  $L$ , including update of  $\text{maxgap}(L)$ , is performed in  $\mathcal{O}(1)$  time.

By Lemma 4, all lcp values can be computed in  $\mathcal{O}(nk)$  time. The lists  $L_j$  can be computed easily in total time  $\mathcal{O}(n)$ .  $\square$

The shortest cover of  $T$  of length at most  $\lfloor n/2 \rfloor$  is a prefix of a solid prefix of  $T$  of length  $\lfloor n/2 \rfloor$ . By the assumption made in the beginning of this section,  $T$  has at most  $\sigma^{k/2}$  solid prefixes of length  $\lfloor n/2 \rfloor$ . For each of them we run the *ShortestCover*( $S, L$ ) algorithm with  $L = \{1, \dots, n\}$ . Lemma 7 implies the following result.

**Theorem 8.** *The shortest cover of an  $i$ -string with  $k$  non-solid symbols can be computed in*  $\mathcal{O}(n\sigma^{k/2}k)$  *time.*

### 4 Algorithm Parameterized by $k$

A border-length of  $T$  is called *ambiguous* if there are at least two different solid borders of  $T$  of this length, otherwise it is called unambiguous. A border of  $T$  is called unambiguous if it corresponds to an unambiguous border-length. By Observation 3, there are at most  $k^2$  ambiguous border-lengths. The main idea of this section is to classify potential covers into two categories depending on whether the length is an unambiguous or an ambiguous border length.

Each unambiguous border is uniquely determined by its length. The solution for this case works in  $\mathcal{O}(nk^4)$  time and uses the subroutine from Section 3. As for the second class, the number of ambiguous border-lengths is at most  $k^2$ . Hence, in this case the problem reduces to testing if there is a cover of a given length (this is still quite nontrivial; as we show later, the whole problem is NP-complete). In fact, the main difficulty is caused by the ambiguous borders.

**Covering with Unambiguous Borders.** For an  $i$ -string  $U$  of length  $m$  and a position  $i$  in  $T$  such that  $U \approx T[i..i + m - 1]$ , we define:

$$U \odot i = U[1] \cap T[i], \dots, U[m] \cap T[i + m - 1].$$

Note that, for a prefix  $U$  of  $T$ , any  $i$ -string of the form  $U \odot i$  can be represented in  $\mathcal{O}(k)$  space (we only store the positions corresponding to non-solid symbols of  $U$ ). Also every solid prefix of  $T$  has such a small representation. We call this a *sparse representation*.

*Example 9.* Let  $T = bb\blacklozenge\blacklozenge abb\blacklozenge\blacklozenge baa$  and  $U = b\blacklozenge a\blacklozenge$ . Then

$$U \odot 1 = U \odot 6 = bba\blacklozenge, U \odot 2 = b\blacklozenge aa, U \odot 7 = b\blacklozenge ab, \text{ and } U \odot 9 = bbaa.$$

The sparse representations of these  $i$ -strings are  $(b, \blacklozenge)$ ,  $(\blacklozenge, a)$ ,  $(\blacklozenge, b)$  and  $(b, a)$ .

A technical modification of the algorithm *ShortestCover* is required to show the following lemma. We omit its full proof in this version of the paper.

**Lemma 10.** *Let  $\mathcal{C}$  be a collection of pairs  $(S, L)$ , where each  $S$  is a solid prefix of  $T$  given in the sparse representation and each  $L$  is an increasing list of positions in  $T$ . If  $|\mathcal{C}| \leq n$  and  $\sum_{(S,L) \in \mathcal{C}} |L| = \mathcal{O}(nk^2)$  then  $ShortestCover(S, L)$  for all instances  $(S, L) \in \mathcal{C}$  can be computed in  $\mathcal{O}(nk^3)$  time.*

By  $SolidOcc(U, T)$  ( $NonSolidOcc(U, T)$ ) we denote the lists of all occurrences  $i \in Occ(U, T)$  for which  $U \odot i$  is a solid string (is not a solid string, respectively). All occurrences of  $U$  in  $T$  can be found using Corollary 5, and divided into these two sets in  $\mathcal{O}(nk^2)$  time. By Observation 3, if  $U$  is a prefix of  $T$  then  $|NonSolidOcc(U, T)| \leq k^2$ .

*Example 11.* Let  $U = a\blacklozenge$ ,  $T = bb\blacklozenge\blacklozenge abb\blacklozenge\blacklozenge ba\blacklozenge$ . Then

$$SolidOcc(U, T) = \{4, 5, 9\}, NonSolidOcc(U, T) = \{3, 8, 11\}.$$

**Theorem 12.** *The shortest cover being an unambiguous border can be computed in  $\mathcal{O}(nk^4)$  time.*

*Proof.* Let  $T$  be an i-string of length  $n$  and  $p_1 < p_2 < \dots < p_r$  be all non-solid symbols in its first half. Let  $p_0 = 1$  and  $p_{r+1} = \lfloor n/2 \rfloor + 1$ . We divide all border-lengths into disjoint intervals  $[p_j, p_{j+1} - 1]$ , for  $j = 0, \dots, r$ .

Consider the interval  $I = [p_j, p_{j+1} - 1]$  and let  $U = T[1..p_j]$ . We compute the lists  $E = \text{SolidOcc}(U, T)$  and  $H = \text{NonSolidOcc}(U, T)$ . Note that for each unambiguous border-length  $d \in I$  we have  $n - d + 1 \in E$ .

We construct a set  $\mathcal{C}$  of different pairs  $(S, L)$ , where each  $S$  is of the form:

$$S = (U \odot i)T[p_j + 1..p_{j+1} - 1] \quad \text{for } i \in E$$

and  $L$  is the list of occurrences of  $U \odot i$  in  $E$  merged with the list  $H$ . If the shortest cover of  $T$  corresponds to an unambiguous border-length from  $I$ , it will be found in one of  $\text{ShortestCover}(S, L)$  calls for  $(S, L) \in \mathcal{C}$ . Note that the lists  $L$  are disjoint on positions from  $E$  and  $|H| \leq k^2$ . We apply Lemma 10 for  $\mathcal{C}$  to obtain  $\mathcal{O}(nk^3)$  time for one instance  $I, U$ , and  $\mathcal{O}(nk^4)$  time in total.  $\square$

**Covering Using Ambiguous Border-Lengths.** In this section we are searching for a solid cover of  $T$  which matches its given prefix  $U = T[1..m]$ . We introduce the following auxiliary problem.

*Problem 13.* Given an i-string  $T$ , an integer  $m$  and a set  $\mathcal{P} \subseteq \text{Occ}(T[1..m], T)$ , find a solid cover  $S \approx T[1..m]$  with a corresponding covering set  $\mathcal{P}' \subseteq \mathcal{P}$  or state that no such  $S, \mathcal{P}'$  exist.

In Lemma 14 we use the *ShortestCover* algorithm in a very similar way to the proof of Theorem 12. We omit the details.

**Lemma 14.** *Problem 13 can be solved in  $\mathcal{O}(nk^3)$  time if the set  $\mathcal{P}' \cap \text{SolidOcc}(T[1..m], T)$  is non-empty.*

For an integer  $m \in \{1, \dots, \lfloor n/2 \rfloor\}$  and a set of positions  $\mathcal{P}'$ , we introduce an auxiliary operation *TestCover*( $m, \mathcal{P}'$ ) which returns true iff there is a cover of  $T$  of length  $m$  for which  $\mathcal{P}'$  is a covering set. This operation is particularly simple to implement for partial words; see the following lemma.

**Lemma 15.** *After  $\mathcal{O}(2^{2k}k + nk^2)$ -time preprocessing, *TestCover*( $m, \mathcal{P}'$ ) can be implemented in  $\mathcal{O}(|\mathcal{P}'|k)$  time. If  $T$  is a partial word then  $\mathcal{O}(nk^2)$ -time preprocessing suffices.*

*Proof.* First consider the simpler case when  $T$  is a partial word. By definition,  $\mathcal{P}'$  can be a covering set for a cover of length  $m$  if and only if  $1 \in \mathcal{P}'$  and  $\text{maxgap}(\mathcal{P}' \cup \{n + 1\}) \leq m$ . These conditions can be easily checked in  $\mathcal{O}(|\mathcal{P}'|)$  time without any preprocessing.

Now it suffices to check if there is a solid string  $S$  of length  $m$  such that  $T[i..i + m - 1] \approx S$  for all  $i \in \mathcal{P}'$ . After  $\mathcal{O}(nk^2)$ -time preprocessing, we can compute  $\text{lcp}(1, i)$  for all  $i \in \mathcal{P}'$  and check if each of those values is at least  $m$ . If not, then certainly such a string  $S$  does not exist. Otherwise, let the set  $Y$

contain positions of all don't care symbols in  $T[1..m]$ . We need to check, for each  $j \in Y$ , if the set

$$X_j = \{T[i - 1 + j] : i \in \mathcal{P}'\}$$

contains at most one solid symbol. This last step is performed in  $\mathcal{O}(|\mathcal{P}'|k)$  time.

If  $T$  is a general  $i$ -string, the only required change is related to processing the  $X_j$  sets. If a set  $X_j$  contains a solid symbol, then it suffices to check if this symbol matches all the other symbols in this set. Otherwise we need some additional preprocessing.

Let  $Z$  be the set of all non-solid positions in  $T$ . We wish to compute, for each subset of  $Z$ , if there is a single solid symbol matching all the positions in this subset. For this, we first reduce the size of the alphabet. For each solid symbol  $c \in \Sigma$ , we find the subset of  $Z$  which contains this symbol. Note that if for two different solid symbols these subsets are equal, we can remove one of those symbols from the alphabet (just for the preprocessing phase). This way we reduce the alphabet size to at most  $2^k$ . Afterwards we simply consider each subset of  $Z$  and look for a common solid symbol, which takes  $\mathcal{O}(2^{2k}k)$  time.  $\square$

We use Lemma 15 to obtain a solution to Problem 13.

**Lemma 16.** *If  $|\mathcal{P}| \leq k^2$ , Problem 13 can be solved in  $\mathcal{O}(2^{(2n/m) \log |\mathcal{P}|}nk/m + 2^{2k}k + nk^2)$  time or  $\mathcal{O}(2^{(2n/m) \log |\mathcal{P}|}nk/m + nk^2)$  time if  $T$  is a partial word.*

*Proof.* Assume there is a solid string  $S \approx T[1..m]$  for which there exists a covering set  $\mathcal{P}' = \{i_1, \dots, i_r\} \subseteq \mathcal{P}$  in  $T$  and further assume that  $|\mathcal{P}'|$  is minimal. Notice that for each  $j \in \{1, \dots, r-2\}$ ,  $i_{j+2} \geq i_j + m$ . Indeed, otherwise  $\mathcal{P}' \setminus \{i_{j+1}\}$  would also be a covering set for  $S$  in  $T$ . Hence,  $r \leq 2n/m$ .

In the algorithm we choose every subset  $\mathcal{P}' \subseteq \mathcal{P}$  of size at most  $\lfloor 2n/m \rfloor$  and run  $\text{TestCover}(m, \mathcal{P}')$ . By Lemma 15, the whole algorithm works in

$$\begin{aligned} \mathcal{O}\left(2^{2k}k + nk^2 + \sum_{i \leq 2n/m} \binom{|\mathcal{P}'|}{i} ik\right) &= \mathcal{O}\left(2^{2k}k + nk^2 + |\mathcal{P}|^{2n/m} \frac{n}{m} k\right) \\ &= \mathcal{O}\left(2^{(2n/m) \log |\mathcal{P}'|}nk/m + 2^{2k}k + nk^2\right) \end{aligned}$$

time. If  $T$  is a partial word, the  $2^{2k}k$  term can be dropped.  $\square$

**Theorem 17.** *The shortest cover of an  $i$ -string with  $k$  non-solid symbols can be computed in  $\mathcal{O}(2^{\mathcal{O}(k \log k)} + nk^5)$  time.*

*Proof.* As candidates for the length of the shortest cover of a given  $i$ -string  $T$  of length  $n$ , we consider all border-lengths. By Theorem 12, we can consider all unambiguous border-lengths in  $\mathcal{O}(nk^4)$  time. There are at most  $k^2$  ambiguous border-lengths. If the cover of such a length  $m$  has an occurrence in  $\text{SolidOcc}(T[1..m], T)$ , due to Lemma 14 it can be computed in  $\mathcal{O}(nk^3)$  time. Across all lengths this gives  $\mathcal{O}(nk^5)$  time.



Note that, by the pigeonhole principle,  $T$  must contain a solid factor of length at least  $\frac{n-k}{k+1}$ . Thus, if  $m \leq \frac{1}{2} \frac{n-k}{k+1}$ , any cover of length  $m$  must have an occurrence within this factor, and consequently an occurrence in  $SolidOcc(T[1..m], T)$ . Therefore, if the cover has no occurrence in  $SolidOcc(T[1..m], T)$ , we have  $m > \frac{n-k}{2k+2}$ . If  $k \leq \frac{n}{3}$  this concludes that  $m > \frac{n}{3k+3}$  and consequently Lemma 16 for  $\mathcal{P} = NonSolidOcc(T[1..m], T)$  yields an  $\mathcal{O}(2^{\mathcal{O}(k \log k)} + nk^2)$ -time algorithm. Otherwise  $k = \Theta(n)$ . Hence, Lemma 16 applied for  $|\mathcal{P}| \leq k^2$  yields an  $2^{\mathcal{O}(n \log k)} = 2^{\mathcal{O}(k \log k)}$ -time solution.  $\square$

**More Efficient Covering of Partial Words.** The Exponential Time Hypothesis (ETH) [13, 19] asserts that for some  $\varepsilon > 0$  the 3-CNF-SAT problem cannot be solved in  $\mathcal{O}(2^{\varepsilon p})$  time, where  $p$  is the number of variables. By the Sparsification Lemma [14, 19], ETH implies that for some  $\varepsilon > 0$  the 3-CNF-SAT problem cannot be solved in  $\mathcal{O}(2^{\varepsilon(p+m)})$ , and consequently in  $2^{\mathcal{O}(p+m)}$  time, where  $m$  is the number of clauses.

We show an algorithm for covering partial word which is more efficient than the generic algorithm for covering i-string. We also show that, unless ETH is false, our algorithm is not far from optimal.

**Theorem 18.**

- (a) *The shortest cover of a partial word with  $k$  don't care symbols can be computed in  $\mathcal{O}(2^{\mathcal{O}(\sqrt{k} \log k)} + nk^5)$  time.*
- (b) *Unless the Exponential Time Hypothesis is false, there is no  $2^{\mathcal{O}(\sqrt{k})} n^{\mathcal{O}(1)}$ -time algorithm computing the shortest cover of a partial word over binary alphabet.*

*Proof.* (a) We improve the algorithm from the proof of Theorem 17. The only part of that algorithm that does not work in  $\mathcal{O}(nk^5)$  time is searching for a cover of length  $m$  being an ambiguous border-length of  $T$ , having all its occurrences in  $H = NonSolidOcc(T[1..m], T)$ . Recall that  $|H| \leq k^2$ . We solve this part more efficiently for partial word  $T$ .

Let  $U = T[1..m]$ . Let  $\mathcal{P} \subseteq H$  be the set of positions such that  $i \in \mathcal{P}$  if and only if  $U \odot i$  has at most  $\sqrt{k}$  don't care symbols. We consider two cases.

**Case 1:** the cover of length  $m$  has an occurrence  $i \in \mathcal{P}$ . Let  $i_1, \dots, i_r$  be the don't care positions in  $U \odot i$ . Let  $M_1, \dots, M_r$  be the sets of all solid symbols at positions  $i_1, \dots, i_r$  of  $U \odot j$  for  $j \in H$ . If any of the sets  $M_a$  is empty, we insert an arbitrary symbol from  $\Sigma$  to it.

Let us construct all possible strings by inserting symbols from  $M_1, \dots, M_r$  at positions  $i_1, \dots, i_r$  in  $U \odot i$ . For each such solid string  $S$ , we simply compute a list  $L$  of all positions  $j \in H$  such that  $U \odot j \approx S$  and check if  $1 \in L$  and if  $\maxgap(L \cup \{n+1\}) \leq m$ . Since  $r \leq \sqrt{k}$  and  $|M_a| \leq |H| \leq k^2$  for all  $a = 1, \dots, r$ , this shows that Case 1 can be solved in  $\mathcal{O}(k^{2\sqrt{k}+1}) = 2^{\mathcal{O}(\sqrt{k} \log k)}$  time.

**Case 2:** the cover of length  $m$  has all its occurrences in  $H \setminus \mathcal{P}$ . Let us divide  $T$  into  $\lfloor \sqrt{k} \rfloor$  fragments of length at least  $\lfloor n/\sqrt{k} \rfloor$  each. By the pigeonhole principle, at least one of those fragments contains at most  $\lfloor \sqrt{k} \rfloor$  don't care symbols. No

occurrence of the cover may be located totally inside this fragment. Therefore,  $m \geq \lfloor \frac{n}{2\sqrt{k}} \rfloor$ . Lemma 16 solves this case in  $2^{\mathcal{O}(\sqrt{k} \log k)} + \mathcal{O}(nk^2)$  time.

(b) In Section 5 we show that the satisfiability problem (CNF-SAT) with  $p$  variables and  $m$  clauses can be reduced to finding the shortest cover of a binary partial word of length  $n = \mathcal{O}((p + m)^2)$ . Thus, unless ETH is false, the latter problem has no  $2^{o(\sqrt{n})}$ -time solution, i.e., no  $2^{o(\sqrt{k})}n^{\mathcal{O}(1)}$  solution.  $\square$

## 5 Hardness of Covering i-Strings and Partial Words

The negative results obtained for partial words remain valid in the more general setting of the i-strings, so in this section we consider partial words only. We shall prove that the following decision problem is NP-complete.

*Problem 19 (d-COVER IN PARTIAL WORDS).* Given a partial word  $T$  of length  $n$  over an alphabet  $\Sigma$  and an integer  $d$ , decide whether there exists a solid cover  $S$  of  $T$  of length  $d$ .

We will reduce from the CNF-SAT problem. Recall that in this problem we are given a Boolean formula with  $p$  variables and  $m$  clauses,  $C_1 \wedge C_2 \wedge \dots \wedge C_m$ , where each clause  $C_i$  is a disjunction of (positive or negative) literals, and our goal is to check if there exists an interpretation that satisfies the formula. Below we present a problem which is equivalent to the CNF-SAT problem, but more suitable for our proof.

*Problem 20 (UNIVERSAL MISMATCH).* Given binary partial words  $W_1, \dots, W_m$  each of length  $p$ , check if there exists a binary partial word  $V$  of length  $p$  such that  $V \not\approx W_i$  for any  $i$ .

**Observation 21.** *The UNIVERSAL MISMATCH problem is equivalent to the CNF-SAT problem, and consequently it is NP-complete.*

*Example 22.* Consider the formula

$$\phi = (x_1 \vee x_2 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_5)$$

with  $m = 3$  and five variables  $(x_1, x_2, x_3, x_4, x_5)$ . In the corresponding instance of the UNIVERSAL MISMATCH problem, for each clause  $C_i$  we construct a partial word  $W_i$  such that  $W_i[j] = 0$  if  $x_j \in C_i$ ,  $W_i[j] = 1$  if  $\neg x_j \in C_i$ , and  $W_i[j] = \diamond$  otherwise:

$$W_1 = 001\diamond 0, \quad W_2 = 1\diamond\diamond 0\diamond, \quad W_3 = \diamond 10\diamond 1.$$

The interpretations  $(1, 0, 1, 1, 0)$ ,  $(1, 1, 1, 1, 0)$  satisfy  $\phi$ . They correspond to partial words 10110, 11110 and  $1\diamond 110$ , none of which matches any of the partial words  $W_1, W_2, W_3$ .



**Lemma 25.** *Let  $c, c' \in \{0, 1, \diamond\}$ , and let  $X, Y$  be partial words of the same length. Then  $11h(Xc)$  occurs in  $\mu(c'Y)01\diamond\diamond$  if and only if  $c \neq c'$ .*

The reduction described above shows that the  $d$ -COVER IN PARTIAL WORDS problem, restricted to the binary alphabet, is NP-hard. Clearly, this problem also belongs to NP, which yields the main result of this section.

**Theorem 26.** *The  $d$ -COVER IN PARTIAL WORDS problem is NP-complete even for the binary alphabet.*

## 6 Conclusions

We considered the problems of finding the length of the shortest solid cover of an indeterminate string and of a partial word. The main results of the paper are fixed-parameter tractable algorithms for these problems parameterized by  $k$ , that is, the number of non-solid symbols in the input. For the partial word covering problem we obtain an  $\mathcal{O}(2^{\mathcal{O}(\sqrt{k} \log k)} + nk^{\mathcal{O}(1)})$ -time algorithm whereas for covering a general indeterminate string we obtain an  $\mathcal{O}(2^{\mathcal{O}(k \log k)} + nk^{\mathcal{O}(1)})$ -time algorithm. The latter can actually be improved to  $\mathcal{O}(2^{\mathcal{O}(k)} + nk^{\mathcal{O}(1)})$  time by extending the tools used in the proof of Theorem 18. In all our algorithms a shortest cover itself and all the lengths of covers could be computed without increasing the complexity.

One open problem is to determine if the shortest cover of indeterminate strings can be found as fast as the shortest cover of partial words. Another question is to close the complexity gap for the latter problem, considering the lower bound resulting from the Exponential Time Hypothesis, which yields that no  $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ -time solution exists for this problem.

## References

1. Abrahamson, K.R.: Generalized string matching. *SIAM Journal on Computing* **16**(6), 1039–1051 (1987)
2. Antoniou, P., Crochemore, M., Iliopoulos, C.S., Jayasekera, I., Landau, G.M.: Conservative string covering of indeterminate strings. In: Holub, J., Ždárek, J. (eds.) *Prague Stringology Conference 2008*, pp. 108–115. Czech Technical University, Prague (2008)
3. Apostolico, A., Ehrenfeucht, A.: Efficient detection of quasiperiodicities in strings. *Theoretical Computer Science* **119**(2), 247–265 (1993)
4. Apostolico, A., Farach, M., Iliopoulos, C.S.: Optimal superprimitivity testing for strings. *Information Processing Letters* **39**(1), 17–20 (1991)
5. Bari, M.F., Rahman, M.S., Shahriyar, R.: Finding all covers of an indeterminate string in  $\mathcal{O}(n)$  time on average. In: Holub, J., Ždárek, J. (eds.) *Prague Stringology Conference 2009*, pp. 263–271. Czech Technical University, Prague (2009)
6. Blanchet-Sadri, F.: *Algorithmic Combinatorics on Partial Words*. Chapman & Hall/CRC Press, Boca Raton (2008)
7. Breslauer, D.: An on-line string superprimitivity test. *Information Processing Letters* **44**(6), 345–347 (1992)

8. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press (2007)
9. Fischer, M.J., Paterson, M.S.: String matching and other products. In: Karp, R.M. (ed.) Complexity of Computation. SIAM-AMS Proceedings, vol. 7, pp. 113–125. AMS, Providence (1974)
10. Holub, J., Smyth, W.F., Wang, S.: Fast pattern-matching on indeterminate strings. *Journal of Discrete Algorithms* **6**(1), 37–50 (2008)
11. Iliopoulos, C.S., Mohamed, M., Mouchard, L., Perdikuri, K., Smyth, W.F., Tsakalidis, A.K.: String regularities with don't cares. *Nordic Journal of Computing* **10**(1), 40–51 (2003)
12. Iliopoulos, C.S., Moore, D., Park, K.: Covering a string. *Algorithmica* **16**(3), 288–297 (1996)
13. Impagliazzo, R., Paturi, R.: On the complexity of  $k$ -SAT. *Journal of Computer and System Sciences* **62**(2), 367–375 (2001)
14. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? *Journal of Computer and System Sciences* **63**(4), 512–530 (2001)
15. Indyk, P.: Faster algorithms for string matching problems: Matching the convolution bound. In: 39th Annual Symposium on Foundations of Computer Science, pp. 166–173. IEEE Computer Society, Los Alamitos (1998)
16. Kalai, A.: Efficient pattern-matching with don't cares. In: Eppstein, D. (ed.) 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 655–656. SIAM, Philadelphia (2002)
17. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: A linear time algorithm for seeds computation. In: Rabani, Y. (ed.) 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1095–1112. SIAM, Philadelphia (2012)
18. Li, Y., Smyth, W.F.: Computing the cover array in linear time. *Algorithmica* **32**(1), 95–106 (2002)
19. Lokshitanov, D., Marx, D., Saurabh, S.: Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS* **105**, 41–72 (2011)
20. Moore, D., Smyth, W.F.: Computing the covers of a string in linear time. In: Sleator, D.D. (ed.) 5th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 511–515. SIAM, Philadelphia (1994)
21. Muthukrishnan, S., Palem, K.V.: Non-standard stringology: algorithms and complexity. In: 26th Annual ACM Symposium on Theory of Computing, pp. 770–779. ACM, New York (1994)
22. Smyth, W.F., Wang, S.: An adaptive hybrid pattern-matching algorithm on indeterminate strings. *International Journal of Foundations of Computer Science* **20**(6), 985–1004 (2009)

# **Scheduling Algorithms**

# Dynamic Interval Scheduling for Multiple Machines

Alexander Gavruskin<sup>2</sup>, Bakhadyr Khoussainov<sup>1</sup>, Mikhail Kokho<sup>1</sup>(✉),  
and Jiamou Liu<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Auckland,  
Auckland, New Zealand

`bmk@cs.auckland.ac.nz`, `m.kokho@auckland.ac.nz`

<sup>2</sup> School of Computing and Mathematical Sciences, Auckland University  
of Technology, Auckland, New Zealand

`gavruskin@aut.ac.nz`, `jiamou.liu@aut.ac.nz`

**Abstract.** We study the dynamic scheduling problem for jobs with fixed start and end times on multiple machines. The problem is to maintain an optimal schedule under the update operations: insertions and deletions of jobs. Call the period of time in a schedule between two consecutive jobs in a given machine an *idle interval*. We show that for any set of jobs there exists a schedule such that the corresponding set of idle intervals forms a tree under the set-theoretic inclusion. Based on this result, we provide a data structure that updates the optimal schedule in  $O(d + \log n)$  worst-case time, where  $d$  is the depth of the set idle intervals and  $n$  is the number of jobs. Furthermore, we show this bound to be tight for any data structure that maintains a nested schedule.

## 1 Introduction

Imagine an operator in a delivery company with two responsibilities. The first is to provide delivery service to clients who request specific times for delivery. The second is to schedule the requests for the drivers such that conflicting requests are assigned to different drivers. The goal of the operator is to accept all client requests and to use as few drivers as possible. The work becomes harder if clients often cancel their requests or change the delivery times of their requests.

The example above is a basic setup for the interval scheduling problem, one of the well-known problems in the theory of scheduling [10, 11]. Formally, the problem can be described as follows. An interval  $a$  is the usual closed non-empty interval  $[s(a), f(a)]$  on the real line. Two intervals  $a$  and  $b$  *overlap* if  $a \cap b \neq \emptyset$ ; otherwise they are *compatible*. We are given a set  $I$  of  $n$  intervals. A subset  $J \subseteq I$  is *compatible* if the intervals in  $J$  are pairwise compatible. The problem is to partition  $I$  into compatible sets  $S_1, \dots, S_k$  such that  $k$  is as small as possible.

Depending on the context, we view an interval  $a$  as either a process or as a set of real numbers. In the first case,  $s(a)$  and  $f(a)$  are respectively the *start* and the *end* of  $a$ . In the second case,  $s(a)$  and  $f(a)$  are the *left* and *right endpoints* of  $a$ . The partition of  $I$  represent schedules for the machines. The *depth* of  $I$  is the maximal number  $d(I)$  of intervals in  $I$  that contain a common point; it

is the maximal number of processes that pass over any single point on the time line. We linearly order elements in the set  $I$  by their left endpoints. Namely, we set  $a \prec b$  whenever  $s(a) < s(b)$  for all  $a, b \in I$ .

A *scheduling function* for the set of intervals  $I$  is a function  $\sigma : I \rightarrow \{1, \dots, k\}$  such that for any two distinct intervals  $a, b \in I$  if  $\sigma(a) = \sigma(b)$  then  $a$  and  $b$  are compatible. The number  $k$  is called the *size* of the scheduling function. The scheduling function  $\sigma : I \rightarrow \{1, \dots, k\}$  partitions  $I$  into  $k$  *schedules*  $S_1, \dots, S_k$ , where for each  $i \in \{1, \dots, k\}$  we have  $S_i = \{a \in I \mid \sigma(a) = i\}$ . It is easy to see that  $d(I)$  is the smallest size of any scheduling function of  $I$ . This gives us the following definition.

**Definition 1.** We call a scheduling function  $\sigma$  optimal if its size is  $d(I)$ .

We briefly describe two algorithms solving the basic interval scheduling problem. The standard greedy algorithm [7], which we call *Algorithm 1*, finds an optimal scheduling function  $\sigma$  for the given set of intervals  $I$  as follows. It starts with sorting the intervals in order of their starting time. Let  $a_1, a_2, \dots, a_n$  be the listing of the intervals in this order. Schedule  $a_1$  into the first machine, that is, set  $\sigma(a_1) = 1$ . Then, for the given interval  $a_i$  and each  $j < i$ , if  $a_i$  and  $a_j$  overlap, exclude the machine  $\sigma(a_j)$  for  $a_i$ . Schedule  $a_i$  into the first machine  $m$  that has not been excluded for  $a_i$  and set  $\sigma(a_i) = m$ . The correctness proof of this algorithm is an easy induction [7]. The algorithm runs in  $O(n^2)$  time. It is important to observe that this algorithm works in a static context in the sense that the set of intervals  $I$  is given a priori and it is not subject to change.

The second algorithm due to Gupta et al. [5], which we call *Algorithm 2*, computes an optimal schedule in  $O(n \log n)$  time. Gupta et al. also show that *Algorithm 2* is the best possible. In this algorithm, we work with endpoints of the intervals in  $I$ . Let  $p_1, p_2, \dots, p_{2n}$  be endpoints of intervals sorted in increasing order. Scan the endpoints from left to right. For each  $p_j$ , if  $p_j$  is the start of some interval  $a$ , find the first available machine and schedule  $a$  into that machine. Otherwise,  $p_j$  is the end of some interval  $b$ . Therefore, mark the machine  $\sigma(b)$  as available. The correctness of the algorithm can be easily verified. Just as above, this algorithm works in a static context.

**The Problem Setup.** In practical applications, the instance of the interval scheduling problem is often changed by a real-time events, and a previously optimal schedule may become sub-optimal. Examples of real-time events include job cancellation, the arrival of an urgent job, and changes in job processing times. To avoid the repetitive work of rerunning the static algorithm every time when the problem instance has changed, there is a demand for efficient dynamic algorithms for solving the partitioning problem on the changed instances. In this dynamic context, the set of intervals changes through a number of update operations, such as insertion or removal. Thus, the *dynamic interval scheduling problem* asks for maintaining an optimal scheduling function  $\sigma$  for a set  $I$  of closed intervals, subject to the following update operations:

- `insert(a)`: insert an interval  $a$  into the set  $I$
- `delete(a)`: delete an interval  $a$  from the set  $I$  (if it is already there).



**Contribution of the Paper.** There are three main technical contributions of the paper. The first concerns the concept of idle intervals. An interval  $(t_0, t_1)$  is *idle* in a given schedule  $\sigma$  if some machine  $\sigma(k)$  stays idle during the time period from  $t_0$  to  $t_1$ . Intuitively, an idle interval is a place in the schedule where we can insert a new interval if its endpoints are between  $t_0$  and  $t_1$ . Now, call the collection of all idle intervals *nested* if any two idle intervals either have no points in common or one interval is included in the other. Firstly, we prove in Lemma 3 that nested schedules are always optimal. Secondly, we prove in Theorem 1 that there are optimal schedules for which the set of idle intervals is nested. This theorem allows us to represent idle intervals of the schedule as a tree, and perform the update operations through maintaining the idle intervals of the schedule. Here we note that Diedrich et al. use idle intervals in [2], where they call them *gaps*, to approximate algorithms for scheduling with fixed jobs. In [2] idle intervals are static and do not depend on the schedule. On the contrary, we describe how to effectively maintain a dynamic set of idle intervals.

Our second contribution is that we provide an optimal data structure that represents nested schedules and supports insert and delete operations. The data structure and its efficiency is based on Theorem 1. Namely, it maintains the nestedness property of the schedules. Theorem 2 proves that all the update operations run in  $O(d + \log(n))$  in the worst-case. Note that if we naively make *Algorithm 1* or *Algorithm 2* dynamic, the update operations of such algorithms will be significantly slower.

Finally, our third contribution is that we prove in Theorem 4 that the bound  $O(d + \log(n))$  is tight for any data structure representing nested schedules.

**Related Work.** There are many surveys on the interval scheduling problem and its variants, also known as “ $k$ -coloring of intervals”, “channel assignment”, “bandwidth allocation” and many others [10, 11]. Gertsbakh and Stern [4] studied the basic problem of scheduling intervals on unlimited number of identical machines. Arkin and Silverber [1] described and solved a weighted version of the interval scheduling problem. In their work the number of machines is restricted and each job has a value. The goal is to maximize the value of completed jobs. A further generalization of the problem, motivated by maintenance of aircraft, was extensively studied by Kroon, Salomon and Wassenhove [12, 13] and by Kolen and Kroon [8, 9]. In this generalization, each job has a class, and each machine is of specific type. The type of a machine specifies which classes of jobs it can process. Since it was shown in [1] that the problem of scheduling classified jobs is NP-complete, the authors study approximation algorithms. Later, Spieksma [17] studied the question of approximating generalized interval scheduling problem.

## 2 Idle Intervals and Nested Scheduling

### 2.1 Idle Intervals

**Definition 2.** Let  $J = \{a_1, a_2, \dots, a_m\}$  be a compatible set of intervals such that  $a_i \prec a_{i+1}$  for each  $i \in \{1, \dots, m-1\}$ . Define the set of idle intervals of  $J$  as the following set:

$$\text{Idle}(J) = \bigcup_{i=1}^{m-1} \{[f(a_i), s(a_{i+1})]\} \cup \{[-\infty, s(a_1)]\} \cup \{[f(a_m), \infty]\}.$$

Note that an idle interval can start at  $-\infty$  or end at  $\infty$ ; they represent a period when a machine is continuously available before or after some moment of time.

The idea behind considering the set of idle intervals is the following. Let  $\sigma : I \rightarrow \{1, \dots, k\}$  be a scheduling function. Recall that  $\sigma$  partitions  $I$  into the schedules  $S_1, \dots, S_k$ . When we insert a new interval  $a$  into  $I$ , we would like to find a gap in some schedule  $S_i$  that fully covers  $a$ . Similarly, a deletion of an interval  $a$  from  $I$  creates a gap in the schedule  $S_{\sigma(a)}$ . Thus, intuitively the insertion and deletion operations are intimately related to the set of idle intervals of the current schedules  $S_1, \dots, S_k$ . Therefore, we need to have a mechanism that efficiently maintains the idle intervals of the scheduling function  $\sigma$ .

**Definition 3.** *The set of idle intervals of the scheduling function  $\sigma$  is*

$$\text{Idle}(\sigma) = \{[-\infty, \infty]\} \cup \text{Idle}(S_1) \cup \text{Idle}(S_2) \cup \dots \cup \text{Idle}(S_k).$$

The scheduling function  $\sigma$  enumerates the idle intervals. Namely, the *schedule number*  $\sigma(b)$  of the idle interval  $b \in \text{Idle}(\sigma)$  is  $i$  if  $b \in \text{Idle}(S_i)$ , and is  $k + 1$  if  $b = [-\infty, \infty]$ . The next lemma states that the depth of the idle interval set is greater than or equal to the depth of the interval set. The proof is straightforward.

**Lemma 1.** *We have  $d(I) \leq k \leq d(\text{Idle}(\sigma))$ .*

**Definition 4.** *A set  $J$  of intervals is nested if  $[-\infty, \infty] \in J$  and for all  $b_1, b_2 \in J$ , it is either that  $b_1$  covers  $b_2$  or  $b_2$  covers  $b_1$  or  $b_1, b_2$  are compatible.*

Any nested set of intervals  $J$  defines a tree under set-theoretic inclusion  $\subseteq$ . Indeed, here the nodes in the tree are the intervals in  $J$ , and an interval  $b_2$  is a descendent of another interval  $b_1$  if  $b_2 \subset b_1$ . We call this tree the *nested tree* of  $J$  and denote it by  $\text{Nest}(J)$ . We order siblings in  $\text{Nest}(J)$  by the left endpoints of the corresponding intervals. Recall that the *height* of a tree is the maximum number of edges in a path that goes from the root to any leaf.

**Lemma 2.** *For any nested set  $J$  of intervals, the depth of  $J$  equals to the height of the nested tree  $\text{Nest}(J)$ .*

*Proof.* Let  $J$  be a nested set of intervals and  $h$  be the height of  $\text{Nest}(J)$ . To show that  $d(J) \leq h$ , we take a maximal path in  $\text{Nest}(J)$ ,  $b_0, b_1, \dots, b_h$  where  $b_0 = [-\infty, \infty]$ , and  $b_{i+1} \subset b_i$  for all  $i \in \{0, \dots, h - 1\}$ . The starting point  $s(b_h)$  intersects with  $h$  intervals. Hence  $d(J) \leq h$

To show the reverse inequality, take any real number  $x \in \mathbb{R}$  and let  $C$  be the set of intervals in  $J$  that contain  $x$ . Then  $C$  is a nested set as well. In particular,  $C$  contains a sequence  $b_1, b_2, \dots, b_\ell$  where  $b_i \subset b_{i+1}$  for all  $i \in \{1, \dots, \ell\}$ . This sequence defines a single path in the tree  $\text{Nest}(J)$ . Thus  $h \leq d(J)$ . □

In the next subsection we connect idle interval sets with the nested trees.

### 2.2 Nested Scheduling

**Definition 5.** Let  $\sigma$  be a scheduling function of the set of intervals  $I$ . We say that  $\sigma$  is a nested schedule if the set  $\text{Idle}(\sigma)$  of idle intervals is nested.

The next lemma shows the usefulness of the notion of nested schedules. In particular, nested schedules are optimal.

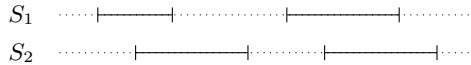
**Lemma 3.** If  $\sigma : I \rightarrow \{1, \dots, k\}$  is a nested scheduling function, then the depth of the idle intervals  $\text{Idle}(\sigma)$  coincides with the depth of  $I$ . In particular, every nested schedule is optimal.

*Proof.* Let  $\sigma : I \rightarrow \{1, \dots, k\}$  be a nested scheduling function for  $I$ . By Lemma 1,  $d(I) \leq d(\text{Idle}(\sigma))$ . To show that  $d(\text{Idle}(\sigma)) \leq d(I)$ , by Lemma 2, it is sufficient to prove that the height  $h$  of the nested tree  $\text{Nest}(\text{Idle}(\sigma))$  is at most  $d(I)$ .

Take a maximal path  $b_0, b_1, \dots, b_h$  in  $\text{Nest}(\text{Idle}(\sigma))$  such that  $f(b_1) \neq \infty$ . For each  $i \in \{1, \dots, h\}$  let  $S_i$  be a schedule such that  $b_i \in \text{Idle}(S_i)$ . We show that in every schedule  $S_i$  there exists an interval  $a_i$  such that  $f(b_1)$  intersects with  $a_i$ .

For contradiction, assume that there exists a schedule  $S_j$  such that  $f(b_1)$  does not intersect with any interval in  $S_j$ . Then there exists an idle interval  $c \in \text{Idle}(S_j)$  such that  $f(b_1) \in c$ . Therefore  $s(c) < f(b_1)$ . On the other hand, since  $b_j \in \text{Idle}(S_j)$ , we have  $s(b_1) < f(b_j) < s(c)$ . These imply that the idle intervals  $b_1$  and  $c$  overlap, which contradicts with the fact that  $\text{Idle}(\sigma)$  is a nested schedule. Thus  $h$  is at most  $d(I)$ . □

A natural question is whether the schedule constructed by either *Algorithm 1* or *Algorithm 2* is nested. Fig. 1 gives a negative answer to this question, where both algorithms yield the same scheduling, which is not nested:



**Fig. 1.** Dotted lines define the idle interval set

### 2.3 Extending Nestedness

We next prove that every interval set  $I$  possesses a nested scheduling. The proof will also provide a way that maintains the interval set  $I$  by keeping the nestedness property invariant under the update operations.

Suppose that  $\sigma : I \rightarrow \{1, \dots, k\}$  is a nested scheduling function for the interval set  $I$ . Recall that we use  $S_1, \dots, S_k$  to denote the  $k$  schedules with respect to  $\sigma$ . Let  $a$  be a new interval not in  $I$ . We introduce the following notations and make several observations to give some intuition to the reader.

- Let  $L \subset \text{Idle}(\sigma)$  be the set of all the idle intervals that contain  $s(a)$ , but do not cover  $a$ . The set  $L$ , as  $\text{Idle}(\sigma)$  is nested, is a sequence of embedded intervals  $x_1 \supset \dots \supset x_\ell$ , where  $\ell \geq 1$ . Note that  $L$  can be the empty set.

- Let  $R \subset \text{Idle}(\sigma)$  be the set of all the idle intervals that contain  $f(a)$ , but do not cover  $a$ . The set  $R$ , as above, is a sequence of embedded intervals  $y_1 \supset \dots \supset y_r$ , where  $r \geq 1$ . Again,  $R$  can be empty as well.
- Let  $z$  be the shortest interval in  $\text{Idle}(\sigma)$  that covers  $a$ . Such an interval exists since  $[-\infty, \infty] \in \text{Idle}(\sigma)$ . To simplify the presentation, we set  $x_0 = y_0 = z$ . Note that  $x_0 \supset x_1$  and  $y_0 \supset y_1$ .

Now our goal is to construct a new nested schedule based on  $\sigma$  and the content of the sets  $L$  and  $R$ . For that we consider several cases.

*Case 1:* The sets  $L$  and  $R$  are empty. In this case we can easily extend  $\sigma$  to the domain  $I \cup \{a\}$  and preserve the nestedness property. Indeed, as  $a \subset z$ , we simply extend  $\sigma$  by setting  $\sigma(a) = \sigma(z)$ . We do not need to change any other schedule. It is not too hard to see that the resulting set of idle intervals is nested.

*Case 2:* The set  $L$  is not empty, but the set  $R = \emptyset$ . We reorganise the schedule  $\sigma$  as follows. We schedule interval  $a$  for the machine  $\sigma(x_\ell)$ . We move all the jobs  $d$  of the machine  $\sigma(x_\ell)$  such that  $d \succ x_\ell$  to machine  $\sigma(x_{\ell-1})$ . We continue this on until we reach the jobs scheduled for the machine  $\sigma(x_1)$ . At this stage, we move all the jobs  $d$  of the machine  $\sigma(x_1)$  such that  $d \succ x_1$  to the machine  $\sigma(z)$ . Recall that  $z = x_0$ . Now, for  $z$  there are two cases. We analyse both.

*Case A:*  $f(z) = +\infty$ . In this case we stop our rescheduling. Denote the resulting schedule by  $\sigma_1$ . Note that if  $z = [-\infty, +\infty]$  then for all  $c$  such that  $\sigma(c) = \sigma(x_1)$  and  $c \succ x_1$  we have  $\sigma_1(c) = k + 1$ . We claim:

*Claim A.* The scheduling  $\sigma_1$  defined is nested.

Indeed,  $\text{Idle}(\sigma_1)$  consists of the new interval  $[f(a), +\infty]$  together with all the idle intervals of  $\sigma$  where the idle intervals  $x_\ell, x_{\ell-1}, \dots, x_1$ , and  $x_0$  are changed to the following new idle intervals  $[s(x_\ell), s(a)]$ ,  $[s(x_{\ell-1}), f(x_\ell)]$ ,  $\dots$ ,  $[s(x_1), f(x_2)]$ , and  $[s(x_0), f(x_1)]$ , respectively. We denote the set of changed intervals by  $L'$ .

Let  $u$  and  $v$  be two idle intervals of  $\sigma_1$ . We want to show that either  $u \cap v = \emptyset$  or one of these two intervals is contained in the other. If both  $u$  and  $v$  are old or both  $u$  and  $v$  are new then we are done. So, say  $u$  is new, and  $v$  is old. First, assume that  $u$  is  $[f(a), +\infty]$ . The interval  $v$  does not contain  $f(a)$  because, by assumption,  $R = \emptyset$ . Therefore, if  $f(a) \leq s(v)$  then  $v \subset u$ ; and if  $s(v) < f(a)$  then  $v \cap u = \emptyset$ . Second, assume  $u$  is  $[s(x_i), f(x_{i+1})]$ , one of the changed intervals. Suppose that  $u$  and  $v$  intersects. If  $v$  contains  $s(a)$  then  $v$  must contain  $x_0$  since  $v$  is old. Hence  $u \subset v$ . Otherwise, suppose that  $v$  contains some point  $r \in u$ . Then either  $r \in x_i$  or  $r \in x_{i+1}$ . Hence,  $v \subset x_i$  or  $v \subset x_{i+1}$ . If the first case we have  $v \subset [s(x_i), s(a)]$ , and in the second case  $v \subset [s(a), f(x_{i+1})]$ . In either case,  $v \subset [s(x_i), f(x_{i+1})]$ . This proves the claim.

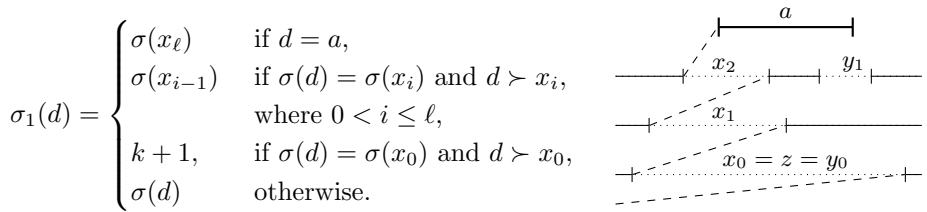
*Case B:*  $f(z) < +\infty$ . Consider  $\sigma_1$  defined above. In this case  $\sigma_1$  might not even be a schedule since the jobs of the machine  $\sigma(z)$  can be incompatible with the jobs of the machine  $\sigma_1(z)$ . So, we change  $\sigma_1$  slightly by moving all the intervals  $d$  of the machine  $\sigma(z)$  such that  $d \succ z$  to the machine  $\sigma(x_\ell)$ . Let us denote the resulting schedule by  $\sigma_2$ .

*Claim B.* The scheduling  $\sigma_2$  defined is nested.

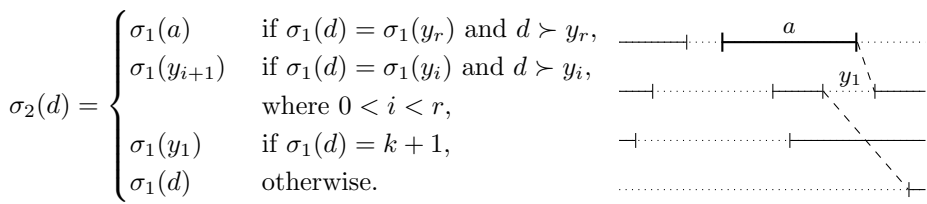
Indeed, one uses the same terminology as in the previous claim. The idle interval  $[f(a), +\infty]$  of  $\sigma_1$  is now replaced with the new idle interval  $[f(a), f(z)]$ . The rest of the proof is the same as the the proof of *Claim A*.

*Case 3:* The set  $R$  is not empty but  $L = \emptyset$ . This case is symmetric to the previous case. So, we leave the details to the reader.

*Case 4:* Both sets  $L$  and  $R$  are non-empty. We reorganize the schedule  $\sigma$  in two steps. In the first step, we proceed exactly as in *Case 2*. Namely, we move all the intervals  $d$  of the machine  $\sigma(x_\ell)$  that start after  $x_\ell$  to  $\sigma(x_{\ell-1})$ ; we continue this by moving all intervals of  $\sigma(x_i)$  that start after  $x_i$  to  $\sigma(x_{i-1})$ . When we reach the machine  $\sigma(x_0)$ , we move all the jobs  $d$  of the machine  $\sigma(x_0)$  such that  $d \succ x_0$  to the machine  $k + 1$ , that is, to the idle interval  $[-\infty, +\infty]$ . Denote the resulting schedule by  $\sigma_1$ . Note that in case  $x_0 = [-\infty, +\infty]$ , no intervals in  $S_{k+1}$  overlap. A formal definition of  $\sigma_1$  is in Fig. 2. In the second step, starting from  $\sigma_1(y_i)$ , where  $i = 1, \dots, r - 1$ , we move all intervals of the machine  $\sigma_1(y_i)$  that start after  $y_i$  to the machine  $\sigma(y_{i+1})$ ; see Fig. 3.



**Fig. 2.** The first step of rescheduling: defining  $\sigma_1$



**Fig. 3.** The second step of rescheduling: defining  $\sigma_2$

**Lemma 4.** *The scheduling function  $\sigma_2$  is nested.*

*Proof.* Let  $K = \{d \in \text{Idle}(\sigma_2) \mid d \subset x_0\}$ . By construction  $\text{Idle}(\sigma_2) \setminus K = \text{Idle}(\sigma) \setminus K$ , and by nestedness of  $\text{Idle}(\sigma)$ ,  $\text{Idle}(\sigma_2) \setminus K$  is also a nested set. Furthermore, it is clear that for any interval  $p \in K$  and  $q \in \text{Idle}(\sigma) \setminus K$ , it is

either that  $p, q$  are compatible or  $p \subset q$ . Therefore it only remains to show that the set  $K$  is also a nested set. We show that any two intervals  $p, q \in K$  are either compatible or one is covered by the other.

Suppose  $p$  contains  $s(a)$ . Then the start of  $p$  is  $s(x_i)$  for some  $0 \leq i \leq \ell$ . Moreover, the end of  $p$  is  $s(a)$ , if  $i = \ell$ , and  $f(x_{i+1})$ , otherwise. Note that  $p \subset x_i$ . Consider two cases with respect to  $q$ :

- Case 1:  $q$  contains  $s(a)$ . Then, similarly to  $p$ , the start of  $q$  is  $s(x_j)$  for some  $0 \leq j \leq \ell$ . If  $x_j \prec x_i$  then  $q$  covers  $p$ . Otherwise,  $p$  covers  $q$ .
- Case 2:  $q$  does not contain  $s(a)$ . Let  $x_j$  be the smallest interval that covers  $q$ . If  $q \prec x_{j+1}$  or  $x_i \prec x_j$  then  $p$  covers  $q$ . Otherwise  $p$  and  $q$  are compatible. In case  $x_j = x_\ell$ , we compare starting times of  $p$  and  $a$ , that is we set  $x_{\ell+1} = a$ .

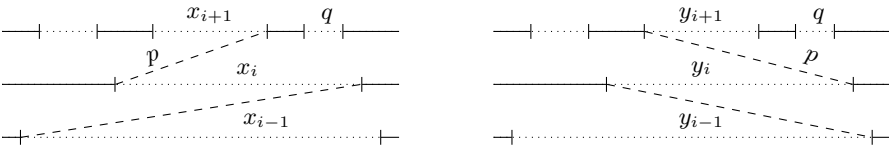


Fig. 4. Nestedness is preserved

Now suppose  $p$  contains  $f(a)$ . Then the end of  $p$  is  $f(y_i)$  for some  $0 \leq i \leq r$ . Moreover, the start of  $p$  is  $f(a)$ , if  $i = r$ , and  $s(y_{i+1})$ , otherwise. Similarly to the previous case, if  $q$  contains  $f(a)$  then, depending on the end of  $q$ , one of the intervals covers the other. If  $q$  does not contain  $f(a)$ , there are two cases:

- Case 1:  $q$  is covered by  $y_r$ . If  $a \prec q$  or  $y_i \prec y_r$  then  $p$  covers  $q$ . Otherwise  $p$  and  $q$  are compatible.
- Case 2:  $p$  is covered by  $y_j$  for some  $0 \leq j < r$  but not covered by  $y_{j+1}$ . If  $y_{i+1} \prec q$  or  $y_i \prec y_j$ , then  $p$  covers  $q$ . Otherwise,  $p$  and  $q$  are compatible.

Finally, suppose that neither  $p$  nor  $q$  contain  $s(a)$  or  $f(a)$ . Then, by construction of  $\sigma_2$ ,  $p$  and  $q$  are in  $\text{Idle}(\sigma)$ . Therefore they are either compatible or one covers the other. Thus the set  $K$  is nested and  $\sigma_2$  is a nested scheduling function.  $\square$

**Theorem 1.** For any set of closed intervals  $I$  there is a scheduling function  $\sigma$  such that  $\text{Idle}(\sigma)$  is a nested set.

*Proof.* We prove by induction on  $|I|$ . When  $|I| = 1$  the statement is clear. The inductive step follows from the construction of  $\sigma_2$  and Lemma 4.  $\square$

### 3 Optimal Data Structure for Nested Scheduling

While various data structures [3,6] can maintain a set of nested intervals, they are not optimal in maintaining the nested schedule. Recall that a nested schedule depends on the set of interval  $I$ . Therefore when we insert or delete an interval, we need to update  $O(d)$  intervals in a nested schedule.

In this section we describe a data structure that takes  $O(d + \log n)$  time for an update operation. Furthermore, we show this bound is tight for any data structure representing nested schedules.

### 3.1 Optimal Data Structure

Our data structure stores idle intervals  $Idle(\sigma)$  that depends on the scheduling function  $\sigma$  and the set of intervals  $I$ . We assume that every endpoint is linked to two corresponding intervals, real and idle (e.g. if a schedule contains intervals  $[3,6]$  and  $[8,9]$ , we store the idle interval  $[6,8]$ ). After each update of  $I$  we restore the nestedness of the idle interval set. Therefore when we insert or delete an interval  $a$ , we update all idle intervals that intersect with the endpoints of  $a$ . Below we describe how to maintain a nested schedule and perform updates.

We store idle intervals in an *interval tree* [14], which is a leaf-oriented binary search tree where leaves store endpoints of the intervals in increasing order. Intervals themselves are stored in the internal nodes as follows. For each internal node  $v$  the set  $I(v)$  consists of intervals that contain the *split point* of  $v$  and are covered by the *range* of  $v$ . The split point of  $v$ , denoted by  $split(v)$ , is a number such that the leaves of the left subtree of  $v$  store endpoints smaller than  $split(v)$ , and the leaves of the right subtree of  $v$  store endpoints greater than  $split(v)$ . The range of  $v$ , denoted by  $range(v)$ , is defined recursively as follows. The range of the root is  $(-\infty, \infty]$ . For a node  $v$ , where  $range(v) = (l, r]$ , the range of the left child of  $v$  is  $(l, split(v)]$ , and the range of the right child of  $v$  is  $(split(v), r]$ . An example of an interval tree is shown in Figure 5.

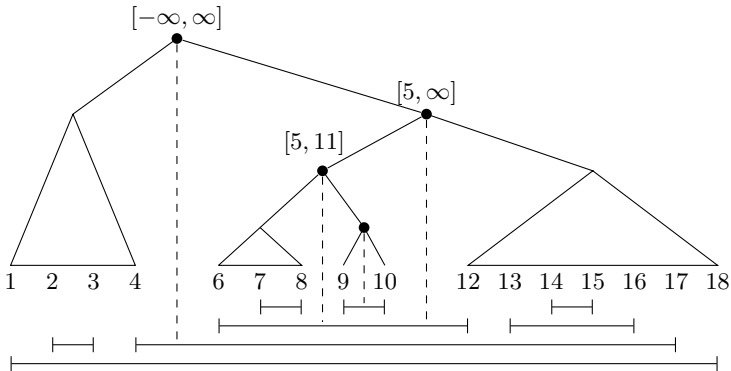


Fig. 5. Nested set of intervals represented by interval tree data structure

Now we describe the update operations on the interval set. Recall that when we insert an interval  $a$ , we need to update idle intervals that intersect with the endpoints of  $a$ . Let  $L$  be the set of idle intervals that contain  $s(a)$ , but not  $f(a)$ . Let  $R$  be the set of idle intervals that contain  $f(a)$ , but not  $s(a)$ . Let  $z$  be the shortest idle interval that contains both endpoints of  $a$ . We show how to update intervals in  $L$ . The update of intervals in  $R$  is similar.

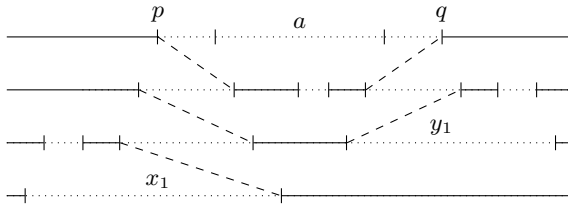
Let  $v_0$  be a node such that  $z \in I(v_0)$ . This node is our starting position. To find intervals in  $L$ , we walk down a path  $v_0, \dots, v_k$  defined by  $s(a)$ . When visiting a node  $v_i$ , we iterate through  $I(v_i)$  and put intervals that contains  $s(a)$

into  $L$ . We delete intervals from  $I(v_i)$  that we put in  $L$ . We stop when we reach a leaf node.

Let  $x_1 \supset \dots \supset x_\ell$  be intervals we have put in  $L$ . We iterate through  $L$  and walk up the path we have traversed. We start iteration from the last interval  $x_\ell$ . For an interval  $x_j$ , we set  $s(x_j) = s(x_{j-1})$ . Then we check if  $x_j$  belongs to  $I(v_i)$ , i.e. if  $split(v_i) \in x_j \subset range(v_i)$ . If  $x_j$  satisfies these conditions, we put  $x_j$  at the beginning of  $I(v_i)$  and remove it from  $L$ . Otherwise, we walk up the path until we find a node with a satisfactory split point and range. Note that no interval in  $I(v_i)$  contains  $s(a)$ , since on the way down we removed all such intervals. Therefore, by the nestedness of idle intervals,  $x_j$  covers all intervals in  $I(v_i)$ .

Finally, we insert  $s(a)$  into the tree. Once inserted, we search for the lowest common ancestor  $v$  of the leaves containing  $s(x_\ell)$  and  $s(a)$ . We add interval  $[s(x_\ell), s(a)]$  into  $I(v)$ .

Deletion of an interval  $a$  is symmetric to insertion. First we delete the endpoints  $s(a)$  and  $f(a)$  and corresponding idle intervals  $[p, s(a)]$  and  $[f(a), q]$  from the interval tree. Now, as the place occupied by  $a$  is free, we have an idle interval  $b = [p, q]$ . An example is shown in Figure 6. The interval  $b$  may violate the nestedness of the idle interval set. Therefore we update idle intervals that intersects with  $b$ . We leave the details of these updates to the reader.



**Fig. 6.** Rescheduling after deletion of the interval  $a$

**Theorem 2.** *The data structure described above maintains the optimal scheduling and supports insertions and deletions in  $O(d + \log n)$  worst-case time.*

*Proof.* When we insert or delete an interval, we update only two sets  $L$  and  $R$  of idle intervals. These two sets corresponds to two paths of length at most  $O(\log n)$ . Furthermore, all intervals in each set share a common point. Therefore the size of each set is at most  $d$ . Since the intervals in internal nodes are ordered, it takes  $O(d)$  time to add intervals into  $L$  and  $R$ . When we put updated intervals back, we add them at the beginning of the lists. Therefore it takes  $O(d)$  time to add intervals from  $L$  and  $R$  into the internal nodes. Finally, we insert or delete at most two leaves. Thus, an update takes  $O(d + \log n)$  time.

The optimality of scheduling after insertion follows from Lemma 4. The optimality of scheduling after deletion can be proved in a similar way.  $\square$



### 3.2 Lower Bound

In this subsection we show that complexity of any data structure that maintains a nested tree is at least  $\Omega(\log n + d)$ , where  $d$  is the height of the nested tree. First we recall a lower bound for the static interval scheduling problem:

**Theorem 3 (Shamos and Hoey [15]).**  *$O(n \log n)$  is a lower bound on the time required to determine if  $n$  intervals on a line are pairwise disjoint.*

**Lemma 5.**  *$O(\log n)$  is a tight bound on the time required to update a data structure that maintains a nested tree.*

*Proof.* For contradiction, assume that there is a data structure with a complexity  $f(n) \in o(\log n)$ . We create a nested tree of  $n$  intervals using this data structure. If the height of the tree is 1, then the intervals do not intersect. However, the time taken is  $n \cdot f(n) \in o(n \log n)$ , which contradicts Theorem 3.  $\square$

**Lemma 6.** *If  $\sigma$  and  $\tau$  are nested scheduling functions then  $\text{Idle}(\sigma) = \text{Idle}(\tau)$ .*

*Proof.* For contradiction, assume that there exist two nested scheduling functions  $\sigma$  and  $\tau$  such that  $\text{Idle}(\sigma) \neq \text{Idle}(\tau)$ . Then there exist two idle intervals  $a_0 \in \text{Idle}(\sigma)$  and  $b_0 \in \text{Idle}(\tau)$  such that they have the same non-infinite starting time, but different finishing times, i.e.  $s(a_0) = s(b_0) \neq -\infty$  and  $f(a_0) \neq f(b_0)$ . Without loss of generality, suppose that  $f(a_0) < f(b_0)$ . Now we take an interval  $b_1$  from  $\text{Idle}(\tau)$  that finishes at  $f(a_0)$ . If its starting time is less than  $s(b_0)$  then intervals  $b_0$  and  $b_1$  overlap, which contradicts the nestedness of  $\tau$ . Otherwise, we continue to  $\text{Idle}(\sigma)$  and take an interval  $a_1$  that starts at  $s(b_1)$ . If  $f(a_1) > f(a_0)$  then  $a_1$  and  $a_0$  overlap and it is a contradiction. Otherwise, we continue in the same manner to  $\text{Idle}(\tau)$ . Since  $I$  is finite, this process eventually stops and one of the scheduling functions appears to be not nested.  $\square$

**Theorem 4.** *An update operation in a data structure representing a nested tree takes at least  $\Omega(\log n + d)$  time.*

*Proof.* Let  $I$  be an interval set and  $\text{Nest}(I)$  be the nested tree of  $I$ . By Lemma 6,  $\text{Nest}(I)$  is unique. Let  $v_0 v_1 \dots v_d$  be longest path in  $\text{Nest}(I)$ . Now consider an interval  $a$ , which starts in the middle of  $v_d$  and finishes after the end of  $v_1$ . Clearly,  $s(a)$  intersects with exactly  $d$  idle intervals. Therefore the trees  $\text{Nest}(I)$  and  $\text{Nest}(I \cup a)$  differ in  $\Omega(d)$  nodes. Taking into account Lemma 5, an update operation of a nested tree requires  $\Omega(\log n + d)$  time.  $\square$

## References

1. Arkin, E.M., Silverberg, E.B.: Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics* **18**(1), 1–8 (1987)
2. Diedrich, F., Jansen, K., Pradel, L., Schwarz, U.M., Svensson, O.: Tight approximation algorithms for scheduling with fixed jobs and nonavailability. *ACM Transactions on Algorithms (TALG)* **8**(3), 27 (2012)

3. Gavruskin, A., Khoussainov, B., Kokho, M., Liu, J.: Dynamising Interval Scheduling: The Monotonic Case. In: Lecroq, T., Mouchard, L. (eds.) IWOCA 2013. LNCS, vol. 8288, pp. 178–191. Springer, Heidelberg (2013)
4. Gertsbakh, I., Stern, H.I.: Minimal resources for fixed and variable job schedules. *Operations Research* **26**(1), 68–85 (1978)
5. Gupta, U.I., Lee, D.T., Leung, J.T.: An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers* **100**(11), 807–810 (1979)
6. Kaplan, H., Molad, E., Tarjan, R.E.: Dynamic rectangular intersection with priorities. In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pp. 639–648. ACM (2003)
7. Kleinberg, J., Tardos, E.: *Algorithm design*. Pearson Education India (2006)
8. Kolen, A.W., Kroon, L.G.: On the computational complexity of (maximum) class scheduling. *European Journal of Operational Research* **54**(1), 23–38 (1991)
9. Kolen, A.W., Kroon, L.G.: An analysis of shift class design problems. *European Journal of Operational Research* **79**(3), 417–430 (1994)
10. Kolen, A.W., Lenstra, J.K., Papadimitriou, C.H., Spieksma, F.C.: Interval scheduling: A survey. *Naval Research Logistics (NRL)* **54**(5), 530–543 (2007)
11. Kovalyov, M.Y., Ng, C.T., Cheng, T.C.: Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research* **178**(2), 331–342 (2007)
12. Kroon, L.G., Salomon, M., Van Wassenhove, L.N.: Exact and approximation algorithms for the operational fixed interval scheduling problem. *European Journal of Operational Research* **82**(1), 190–205 (1995)
13. Kroon, L.G., Salomon, M., Van Wassenhove, L.N.: Exact and approximation algorithms for the tactical fixed interval scheduling problem. *Operations Research* **45**(4), 624–638 (1997)
14. Mehlhorn, K.: *Data structures and algorithms. Multi-dimensional Searching and Computational Geometry*, vol. 3. Springer, Berlin (1984)
15. Shamos, M.I., Hoey, D.: Geometric intersection problems. In: *17th Annual Symposium on Foundations of Computer Science*, pp. 208–215. IEEE (1976)
16. Sleator, D., Tarjan, R.: A Data Structure for Dynamic Trees. *Journal of Computer and System Sciences* **26**(3), 362–391 (1983)
17. Spieksma, F.C.: On the approximability of an interval scheduling problem. *Journal of Scheduling* **2**(5), 215–227 (1999)

# Throughput Maximization in Multiprocessor Speed-Scaling

Eric Angel<sup>1</sup>, Evripidis Bampis<sup>2</sup>, Vincent Chau<sup>1(✉)</sup>, and Nguyen Kim Thang<sup>1</sup>

<sup>1</sup> IBISC, Université d'Évry Val d'Essonne, Évry, France  
vincent.chau@ibisc.univ-evry.fr

<sup>2</sup> Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6 Paris, France

**Abstract.** In the classical energy minimization problem, introduced in [Yao et al., FOCS'95], we are given a set of  $n$  jobs each one characterized by its release date, its deadline, its processing volume and we aim to find a feasible schedule of the jobs on a single speed-scalable machine so that the total energy consumption is minimized. Here, we study the *throughput maximization* version of the problem where we are given a budget of energy  $E$  and where every job has also a value. Our goal is to determine a feasible schedule maximizing the (weighted) throughput of the jobs that are executed between their respective release dates and deadlines. We first consider the preemptive non-migratory multiprocessor case in a *fully heterogeneous* environment in which every job has a machine-dependent release date, deadline and processing volume and every machine obeys to a different speed-to-power function. We present a polynomial time greedy algorithm based on the primal-dual scheme that approximates the optimum solution within a factor depending on the energy functions (the factor is constant for typical energy functions of form  $P(z) = z^\alpha$ ). Then, we focus on the *non-preemptive* case for which we consider a *fixed* number of identical parallel machines and two important families of instances: (1) equal processing volume jobs; and (2) agreeable jobs. For both cases we present optimal pseudo-polynomial-time algorithms.

## 1 Introduction

Power management has become a major issue in our days. One of the mechanisms used for saving energy in computing systems is speed-scaling where the speed of the machines can dynamically change over time. We adopt the model first introduced by Yao et al. [24] and we study the multiprocessor scheduling problem of maximizing the throughput of jobs for a given budget of energy. Maximizing throughput, i.e. the number of jobs or the total weight of jobs executed on time for a given budget of energy, is a very natural objective in this

---

Research supported by FMJH program Gaspard Monge in Optimization and Operations Research and by EDF, by the project PHC CAI YUANPEI (27927VE) and by the project ALGONOW, co-financed by the European Union (European Social Fund - ESF).

setting. Indeed mobile devices, such as mobile phones or computers, have a limited energy capacity depending on the quality of their battery, and throughput is one of the most popular objectives in scheduling literature for evaluating the performance of scheduling algorithms for problems involving jobs that are subject to release dates and deadlines [13, 22]. Different variants of the throughput maximization problem in the online speed-scaling setting have been studied in the literature [12, 15, 16, 21, 23]. However, in the off-line context, only recently, an optimal pseudopolynomial-time algorithm has been proposed for *preemptive* scheduling (i.e., the execution of a job may be interrupted and resumed later) on a single machine [4]. Up to our knowledge no results are known for the throughput maximization problem in the multiprocessor case. In this paper, we address this issue. More specifically, we first consider the case of a set of parallel machines in a fully heterogeneous environment [10]. Every job has a value, a machine-dependent release date, deadline and processing volume and every machine obeys to a different speed-to-power function. We propose a polynomial-time constant factor approximation algorithm for the problem of maximizing the weighted throughput in the *preemptive non-migratory* setting (i.e, a job must be executed entirely on at most one machine). Our algorithm combines the use of the knapsack inequalities, presented in [14], in order to minimize the integrality gap of the relaxation and the use of a primal-dual scheme on a linearized version of a convex program with linear constraints that is inspired by the approach used in [18] for the online matching problem. In the second part of the paper, we study the *non-preemptive* throughput maximization problem. Given that the non-preemptive energy minimization problem has been recently proved APX-hard for the heterogeneous model [17] and that no constant factor approximation is known for the identical machines case. We focus on the throughput maximization problem with a *fixed* number of identical parallel machines for two important families of instances. More precisely, we study the case where all the jobs have the same processing volume, and the case where the jobs have arbitrary processing volumes, but their release dates and deadlines are agreeable where earlier released jobs have earlier deadlines. We propose exact algorithms that are based on a discretization of the problem and the use of dynamic programming.

*Problem Definition and Notations.* In the first part of the paper, we consider the problem in a fully heterogeneous environment. Formally, there are  $m$  parallel machines and  $n$  jobs. Each job  $j$  has its release date  $r_{ij}$ , deadline  $d_{ij}$ , and its processing volume  $p_{ij}$  on machine  $i$ . Moreover, job  $j$  has weight  $w_j$  which represents its value. If a job is executed on machine  $i$  then it must be entirely processed during its available time interval  $[r_{ij}, d_{ij}]$  on that machine without migration. The *weighted throughput* of a schedule is  $\sum_{i,j} w_j$  where the sum is taken over jobs  $j$  completed on machine  $i$ . At any time, a machine can choose a speed to process a job. If the speed of machine  $i$  at time  $t$  is  $s_i(t)$  then the energy power at  $t$  is  $P_i(s_i(t))$  where  $P_i$  is a given convex power energy function of machine  $i$ . Typically, one has  $P_i(z) := z^{\alpha_i}$  where  $2 \leq \alpha_i \leq 3$ . The *consumed energy* on machine  $i$  is  $\int_0^\infty P_i(s_i(t))dt$ . Our objective is to maximize the weighted throughput for a given budget of energy  $E$ . Hence, the scheduler has to decide

the set of jobs that will be executed, assign the jobs to machines and choose appropriate speeds to schedule these jobs without exceeding the energy budget.

In the second part of the paper we consider identical parallel machines where the job parameters (i.e., its release date, deadline, processing volume) are not machine-dependent and the energy power functions are the same for every machine, e.g.,  $P(z) = z^\alpha$ . We study the problem for two families of instances: (1) instances with identical processing volumes, i.e.  $p_{i,j} = p$  for every  $i$  and  $j$ , and (2) agreeable instances, i.e. instances for which  $r_i \leq r_j$  if and only if  $d_i \leq d_j$ .

In the sequel, we need the following definition: Given arbitrary convex functions  $P_i$  as the energy power functions, define  $\Gamma := \max_i \max_{z>0} zP'_i(z)/P_i(z)$  (a parameter of function  $P$ ). As said before, for the most studied case in the literature one has  $P_i(z) = z^{\alpha_i}$ , and therefore  $\Gamma = \max_i \alpha_i$  is constant.

## 1.1 Related Work

Many papers considered the closely related problem of minimizing the consumed energy. For the *preemptive single-machine case*, Yao et al. [24] in their seminal paper proposed an optimal polynomial-time algorithm. Since then, a lot of papers appeared in the literature (for a survey see [1]). Antoniadis and Huang [7] have considered the non-preemptive energy minimization problem. They proved that the *non-preemptive single-machine case* is strongly NP-hard and they proposed an approximation algorithm. This result has been improved recently in [10] and [17]. For instances in which all the jobs have the same processing volume, a  $2^\alpha$ -approximation for the single-machine case has been presented in [9]. However the complexity status of this problem remained open. In this paper, we settle this question even for the identical machine case where the number of machines is a fixed constant. Notice that independently, Huang et al. in [20] proposed a polynomial-time algorithm for the single machine case.

The *multiple machine case* where the *preemption and the migration* of jobs are allowed can be solved in polynomial time [2,6,11]. For the *heterogeneous multiprocessor speed-scaling problem* with preemptions and migrations allowed, an algorithm that returns a solution which is within an additive factor of  $\epsilon$  far from the optimal solution and runs in time polynomial to the size of the instance and to  $\frac{1}{\epsilon}$  has been proposed in [10]. Albers et al. [3] considered the multiple machine problem where the *preemption of jobs* is allowed but *not their migration*. They provided numerous exact and approximation algorithms for instances with equal processing volumes and/or agreeable instances. Their approximation ratio for agreeable instances has been improved to  $(2 - 1/m)^{\alpha-1}$  in [9]. Greiner et al. [19] proposed a  $B_{\lceil \alpha \rceil}$ -approximation algorithm for general instances, where  $B_{\lceil \alpha \rceil}$  is the  $\alpha$ -th Bell number. For the *heterogeneous multiprocessor speed-scaling problem* with preemption but no migrations, an approximation algorithm of ratio  $(1 + \epsilon)\tilde{B}_\alpha$  where  $\tilde{B}_\alpha$  is the generalized Bell number, has been proposed in [10]. For the *non-preemptive multiple machine energy minimization problem*, a  $2(1 + \epsilon)(5(1 + \epsilon))^{\alpha-1}\tilde{B}_\alpha$ -approximation algorithm have been proposed in [17] for the case where all the processing volumes of the jobs are the same. For arbitrary machine-dependent processing volumes, Cohen-Addad et al. in [17]

proved that the problem becomes APX-hard and they proposed a non-constant factor approximation algorithm with an approximation factor that depends on the maximum ratio between two processing volumes.

Angel et al. studied the *throughput maximization* problem in the offline setting in [5]. They provided a polynomial time algorithm to solve optimally the *single-machine* problem for *agreeable* instances. More recently in [4], they proved that there is a pseudo-polynomial time algorithm for solving optimally the *pre-emptive single-machine* problem with arbitrary release dates and deadlines and arbitrary processing volumes. For the weighted version, the problem is  $\mathcal{NP}$ -hard even for instances in which all the jobs have common release dates and deadlines. Angel et al. [5] showed that the problem admits a pseudo-polynomial time algorithm for agreeable instances. Furthermore, Antoniadis et al. [8] considered a generalization of the classical knapsack problem where the objective is to maximize the total profit of the chosen items minus the cost incurred by their total weight. The case where the cost functions are convex can be translated in terms of a weighted throughput problem where the objective is to select the most profitable set of jobs taking into account the energy costs. Antoniadis et al. presented a FPTAS and a fast 2-approximation algorithm for the non-preemptive problem where the jobs have no release dates or deadlines.

Up to the best of our knowledge, no work is known for the offline throughput maximization problem in the case of multiple machines.

## 1.2 Our Approach and Contributions

In Section 2, we consider the throughput maximization problem of scheduling a set of  $n$  jobs on  $m$  parallel speed-scalable machines in a fully *heterogeneous environment*. Every job has a machine-dependent release date, deadline and processing volume and every machine obeys to a different speed-to-power function. Instead of studying the problem directly, we study the related problem of minimizing the consumed energy under the constraint that the total weighted throughput must be at least some given throughput demand  $W$ .

For the problem of minimizing the energy's consumption under the throughput constraint, we present a polynomial time algorithm which has the following property: the consumed energy of the algorithm given a throughput demand  $W$  is at most that of an optimal schedule with throughput demand  $2(\Gamma + 1)W$ . The algorithm is based on a primal-dual scheme for mathematical programs with linear constraints and a convex objective function. Specifically, our approach consists in considering a relaxation with convex objective and linear constraints. Then, we linearize the convex objective function and construct a dual program. Using this procedure, the strong duality is not necessarily ensured but the weak duality always holds and that is indeed the property that we need for our approximation algorithm. The linearization and the dual construction follow the scheme introduced in [18] for online matching. In the relaxation, we also make use of the knapsack inequalities, presented in [14], in order to reduce the integrality gap in the multiprocessor environments. The algorithm follows the standard primal-dual framework: at any time, some dual variables are greedily

increased until some dual constraint becomes tight. Then some job is selected and is dispatched to the corresponding machine revealed by the dual constraint. Typically, one will bound the primal objective value by the dual one. In the analysis, instead of comparing the primal and its dual, we bound the primal by the dual of the original relaxation but with the new demand which is  $2(\Gamma + 1)$  times larger. An advantage in the analysis is that the feasible solutions of the dual program corresponding to the primal with demand  $W$  is also feasible for the dual corresponding to the primal with demand  $2(\Gamma + 1)W$ .

For the problem of maximizing the throughput under a given budget of energy, we apply a dichotomy search using as subroutine the algorithm for the problem of minimizing the energy's consumption with a given weighted throughput demand. Our algorithm is a  $2(\Gamma + 1)(1 + \epsilon)$ -approximation for the weighted throughput where  $\epsilon > 0$  is an arbitrarily small constant. The algorithm's running time is polynomial in the input size of the problem and  $1/\epsilon$ . Clearly, one may be interested in finding a tradeoff between the precision and the running time of the algorithm.

Given that the non-preemptive energy minimization problem is proved APX-hard for the heterogeneous model [17] and that no constant factor approximation is known for the identical machines case, in Section 3, we focus on the throughput maximization problem with a *fixed* number of *identical parallel* machines for two important families of instances. By identical machines, we mean that  $p_{i,j} = p_j$ , i.e. the processing volume of every job is independent of the machine on which it will be executed. Moreover,  $r_{ij} = r_j$  and  $d_{ij} = d_j$  for every job  $j$  and every machine  $i$ . The problem is weakly  $\mathcal{NP}$ -hard even in a very restricted special case in which there is a single machine, jobs have equal processing volume and the release dates and deadlines are *agreeable* (for every jobs  $j$  and  $j'$ , if  $r_j < r_{j'}$  then  $d_j \leq d_{j'}$ ). Hence, we focus on two cases: (1) jobs that have the same processing volumes, but arbitrary release dates and deadlines; and (2) jobs that have arbitrary processing volumes, but their release dates and deadlines are agreeable. We present pseudo-polynomial time algorithms based on dynamic programming for these variants. Specifically, when all jobs have the same processing volume, our algorithm has running time  $O(n^{12m+7}W^2)$  where  $W = \sum_j w_j$ . Note that when jobs have unit weight, the algorithm has polynomial running time. When jobs are agreeable, our algorithm has running time  $O(n^{2m+2}V^{2m+1}Wm)$  where  $V = \sum_j p_j$ . Using standard techniques, these algorithms may lead to approximation schemes.

## 2 Approximation for Non-Migratory Scheduling

We first study a related problem in which we look for an algorithm that minimizes the consumed energy under the constraint of throughput demand. Then, we use that algorithm as a sub-routine to derive an algorithm for the problem of maximizing throughput under the energy constraint.

*Energy Minimization with Throughput Demand Constraint.* In the problem, there are  $n$  jobs and  $m$  parallel machines. A job  $j$  has release date  $r_{ij}$ , deadline

$d_{ij}$ , weight  $w_j$  and processing volume  $p_{ij}$  if it is scheduled on machine  $i$ . Given a throughput demand of  $W$ , the scheduler needs to choose a subset of jobs, assign them to the machines and decide the speed to process these jobs in such a way that the total weight (throughput) of completed jobs is at least  $W$  and the consumed energy is minimized. Jobs are allowed to be processed preemptively, but without migration.

Let  $x_{ij}$  be a variable indicating whether job  $j$  is scheduled on machine  $i$ . Let  $s_{ij}(t)$  be a variable representing the speed that the machine  $i$  uses in order to process job  $j$  at time  $t$ . The problem can be formulated as the primal convex relaxation  $(\mathcal{P})$ .

$$\begin{aligned}
 \min \quad & \sum_i \int_0^\infty P_i(s_i(t)) dt & (\mathcal{P}) \\
 \text{subject to} \quad & s_i(t) = \sum_j s_{ij}(t) & \forall i, t \\
 & \sum_i x_{ij} \leq 1 & \forall j \quad (1) \\
 & \int_{r_{ij}}^{d_{ij}} s_{ij}(t) dt \geq p_{ij} x_{ij} & \forall i, j \quad (2) \\
 & \sum_i \sum_{j:j \notin S} w_j^S x_{ij} \geq W - w(S) \quad \forall S \subset \{1, \dots, n\} & (3) \\
 & x_{ij}, s_{ij}(t) \geq 0 & \forall i, j, t
 \end{aligned}$$

In the formulation, constraints (1) ensure that a job can be chosen at most once. Constraints (2) guarantee that job  $j$  must be completed if it is assigned to machine  $i$ . To satisfy the throughput demand constraint, we use the knapsack inequalities (3) introduced in [14]. Note that in the constraints,  $S$  is a subset of jobs,  $w(S) = \sum_{j \in S} w_j$  and  $w_j^S := \min\{w_j, W - w(S)\}$ . Intuitively, if  $S$  is the set of jobs which will be completed then one need to cover  $W - w(S)$  amount of throughput over jobs not in  $S$  in order to satisfy the demand. These constraints reduce significantly the integrality gap of the relaxation compared to the natural constraint  $\sum_{ij} w_j x_{ij} \geq W$ .

Observe that the primal consist of linear constraints and a convex objective function. Hence, the idea of the approach is to derive a closed form dual program which is intuitive in the sense of linear programming by linearizing the objective function. Define functions  $Q_i(z) := P_i(z) - zP'_i(z)$  for every machine  $i$ . Consider the following dual program  $(\mathcal{D})$ .

The construction of the dual  $(\mathcal{D})$  is inspired by [18] and is obtained by linearizing the convex objective of the primal. By this procedure the strong duality is not necessarily guaranteed, but the weak duality always holds. Indeed we only need the weak duality for approximation algorithms. In fact, the dual  $(\mathcal{D})$  gives a meaningful lower bound that we will exploit to design our approximation algorithm.



$$\max \quad \sum_S (W - w(S))\beta_S + \sum_i \int_0^\infty Q_i(v_i(t))dt - \sum_j \gamma_j \quad (\mathcal{D})$$

$$\text{s.t.} \quad \lambda_{ij} \leq P'_i(v_i(t)) \quad \forall i, j, \forall t \in [r_{ij}, d_{ij}] \quad (4)$$

$$\sum_{S: j \notin S} w_j^S \beta_S \leq \gamma_j + \lambda_{ij} p_{ij} \quad \forall i, j \quad (5)$$

$$\beta_S, \lambda_{ij}, \gamma_j, v_i(t) \geq 0 \quad \forall i, j, \forall t, \forall S \subset \{1, \dots, n\}$$

**Lemma 1 (Weak Duality).** *The optimal value of the dual program (D) is at most the optimal value of the primal program (P).*

The primal/dual programs (P) and (D) highlight the main ideas of the algorithm. Intuitively, if a job  $j$  is assigned to machine  $i$  then one must increase the speed of job  $j$  on machine  $i$  at  $\arg \min P'_i(v_i(t))$  in order to always satisfy the constraint (4). Moreover, when constraint (5) becomes tight for some job  $j$  and machine  $i$ , one could assign  $j$  to  $i$  in order to continue to raise some  $\beta_S$  and increase the dual objective. The formal algorithm is given below.

---

**Algorithm 1.** Minimizing the consumed energy under the throughput constraint

---

- 1: Initially, set  $s_i(t), s_{ij}(t), v_i(t)$  and  $\lambda_{ij}, \gamma_j$  equal to 0 for every job  $j$ , machine  $i$  and time  $t$ .
  - 2: Initially,  $\mathcal{T} \leftarrow \emptyset$ .
  - 3: **while**  $W > w(\mathcal{T})$  **do**
  - 4:   **for** every job  $j \notin \mathcal{T}$  and every machine  $i$  **do**
  - 5:     Continuously increase  $s_{ij}(t)$  at  $\arg \min P'_i(v_i(t))$  for  $r_{ij} \leq t \leq d_{ij}$  and simultaneously update  $v_i(t) \leftarrow v_i(t) + s_{ij}(t)$  until  $\int_{r_{ij}}^{d_{ij}} s_{ij}(t)dt = p_{ij}$ .
  - 6:     Set  $\lambda_{ij} \leftarrow \min_{r_{ij} \leq t \leq d_{ij}} P'_i(v_i(t))$ .
  - 7:     Reset  $v_i(t)$  as before, i.e.,  $v_i(t) \leftarrow v_i(t) - s_{ij}(t)$  for every  $t \in [r_{ij}, d_{ij}]$ .
  - 8:   **end for**
  - 9:   Continuously increase  $\beta_{\mathcal{T}}$  until  $\sum_{S: j \notin S} w_j^S \beta_S = p_{ij} \lambda_{ij}$  for some job  $j$  and machine  $i$ .
  - 10:   Assign job  $j$  to machine  $i$ . Set  $s_i(t) \leftarrow s_i(t) + s_{ij}(t)$  and  $v_i(t) \leftarrow s_i(t)$  for every  $t$ .
  - 11:   Set  $\mathcal{T} \leftarrow \mathcal{T} \cup \{j\}$ . Moreover, set  $\gamma_j \leftarrow p_{ij} \lambda_{ij}$ .
  - 12:   Reset  $\lambda_{i'j} \leftarrow 0$  and  $s_{i'j}(t) \leftarrow 0$  for every  $i' \neq i$ .
  - 13: **end while**
- 

In the algorithm  $\arg \min P'_i(v_i(t))$  for  $r_{ij} \leq t \leq d_{ij}$  is defined as  $\{t : t \in [r_{ij}, d_{ij}] \text{ and } P'_i(v_i(t)) = \min_{r_{ij} \leq x \leq d_{ij}} P'_i(v_i(x))\}$ , this is usually a set of intervals, and thus the speed  $s_{ij}$  is increased simultaneously on a set of intervals. Notice also that since  $P_i$  is a convex function,  $P'_i$  is non decreasing. Hence, in line 5 of the algorithm,  $\arg \min P'_i(v_i(t))$  can be replaced by  $\arg \min v_i(t)$ ; so we can avoid the computation of the derivative  $P'_i(z)$ . Note that at the end of the algorithm variables  $v_i(t)$  are indeed equal to  $s_i(t)$  — the speed of machine  $i$  for every  $i$ .

Given the assignment of jobs and the speed function  $s_i(t)$  of each machine  $i$  returned by the algorithm, one can process the jobs on each machine using the Earliest Deadline First (EDF) order.

**Lemma 2.** *The solution  $\beta_S, \gamma_j$  and  $v_i(t)$  for every  $i, j, S, t$  constructed by Algorithm 1 is feasible for the dual ( $\mathcal{D}$ ).*

**Theorem 1.** *The consumed energy of the schedule returned by the algorithm 1 with a throughput demand of  $W$  is at most the energy of the optimal schedule with a throughput demand  $2(\Gamma + 1)W$ .*

*Proof.* Let  $OPT(2(\Gamma + 1)W)$  be the energy consumed by the optimal schedule with the throughput demand  $2(\Gamma + 1)W$ . By Lemma 1, we have that:

$OPT(2(\Gamma + 1)W) \geq \sum_S (2(\Gamma + 1)W - w(S))\beta_S + \sum_i \int_0^\infty Q_i(v_i(t))dt - \sum_j \gamma_j$  where the variables  $\beta_S, v_i, \gamma_j$  satisfy the same constraints in the dual ( $\mathcal{D}$ ). Therefore, it is sufficient to prove that latter quantity is larger than the consumed energy of the schedule returned by the algorithm with the throughput demand  $W$ , denoted by  $ALG(W)$ . Specifically, we will prove a stronger claim. For  $\beta_S, \gamma_j$  and  $v_i$  (which is equal to  $s_i$ ) in the feasible dual solution constructed by Algorithm 1 with the throughput demand  $W$ , it always holds that:  $2(\Gamma + 1) \sum_S (W - w(S))\beta_S + \sum_i \int_0^\infty Q_i(s_i(t))dt - \sum_j \gamma_j \geq \sum_i \int_0^\infty P_i(s_i(t))dt$ .

By the algorithm, we have that

$$\sum_{i,j \in \mathcal{T}} p_{ij} \lambda_{ij} = \sum_{j \in \mathcal{T}} \sum_{S: j \notin S} w_j^S \beta_S = \sum_S \beta_S \left( \sum_{j \in \mathcal{T} \setminus S} w_j^S \right) \leq 2 \sum_S \beta_S (W - w(S)) \tag{6}$$

In the second sum,  $\beta_S \neq 0$  iff  $S$  equals  $\mathcal{T}$  at some step during the execution of the algorithm. Thus, we consider only such sets in that sum. Let  $j^*$  be the last element added to  $\mathcal{T}$ . For  $S \subset \mathcal{T} \setminus \{j^*\}$  and  $\beta_S > 0$ , by the while loop condition  $w(S) + \sum_{j \notin S, j \in \mathcal{T} \setminus \{j^*\}} w_j^S < W$ . Moreover,  $w_{j^*}^S \leq W - w(S)$  by definition. Hence,  $\sum_{j \notin S, j \in \mathcal{T}} w_j^S \leq 2(W - w(S))$  and the inequality (6) follows.

Fix a machine  $i$  and let  $\{1, \dots, k\}$  be the set of jobs assigned to machine  $i$  (renaming jobs if necessary). Let  $u_{i1}(t), \dots, u_{ik}(t)$  be the speed of machine  $i$  at time  $t$  after assigning jobs  $1, \dots, k$ , respectively. In other words,  $u_{i\ell}(t) = \sum_{j=1}^\ell s_{ij}(t)$  for every  $1 \leq \ell \leq k$ . By the algorithm, we have  $\lambda_{i\ell} = \min_{r_\ell \leq t \leq d_\ell} P'_i(u_{i\ell}(t))$  for every  $1 \leq \ell \leq k$ . As every job  $\ell$  is completed in machine  $i$ ,  $\int_{r_\ell}^{d_\ell} s_{i\ell}(t)dt = p_{i\ell}$ . Note that  $s_{i\ell}(t) > 0$  only at  $t$  in  $\arg \min_{r_\ell \leq t \leq d_\ell} P'_i(u_{i\ell}(t))$ . Thus,

$$\begin{aligned} \sum_{\ell=1}^k \lambda_{i\ell} p_{i\ell} &= \sum_{\ell=1}^k \int_{r_\ell}^{d_\ell} s_{i\ell}(t) P'_i \left( \sum_{j=1}^\ell s_{ij}(t) \right) dt = \sum_{\ell=1}^k \int_0^\infty s_{i\ell}(t) P'_i \left( \sum_{j=1}^\ell s_{ij}(t) \right) dt \\ &\geq \sum_{\ell=1}^k \int_0^\infty \left[ P_i \left( \sum_{j=1}^\ell s_{ij}(t) \right) - P_i \left( \sum_{j=1}^{\ell-1} s_{ij}(t) \right) \right] dt \\ &= \int_0^\infty \left[ P_i(u_{ik}(t)) - P_i(0) \right] dt = \int_0^\infty P_i(s_i(t)) dt \end{aligned} \tag{7}$$

where in the second equality, note that  $s_{i\ell}(t) = 0$  for  $t \notin [r_\ell, d_\ell]$ ; the inequality is due to the convexity of  $P_i$ .

As inequality (7) holds for every machine  $i$ , summing over all machines we get  $\sum_{i,j \in \mathcal{T}} p_{ij} \lambda_{ij} \geq \sum_i \int_0^\infty P_i(s_i(t)) dt$ . Together with (6), we deduce that

$$\begin{aligned} & 2(\Gamma + 1) \sum_S \beta_S (W - w(S)) + \sum_i \int_0^\infty Q_i(s_i(t)) dt - \sum_j \gamma_j \\ & \geq \sum_{i,j \in \mathcal{T}} p_{ij} \lambda_{ij} + \Gamma \sum_i \int_0^\infty P_i(s_i(t)) dt + \sum_i \int_0^\infty Q_i(s_i(t)) dt - \sum_j \gamma_j \\ & \geq \sum_i \int_0^\infty P_i(s_i(t)) dt = ALG(W). \end{aligned}$$

where the last inequality is due to the definition of  $\Gamma$  (recall that  $\Gamma = \max_i \max_z z P'_i(z) / P_i(z)$  for every  $z$  such that  $P(z) > 0$ ) and  $\gamma_j = \sum_i \lambda_{ij} p_{ij}$  for every job  $j$  in  $\mathcal{T}$  (by the algorithm).  $\square$

**Corollary 1.** *For the single machine setting, the consumed energy of the schedule returned by the algorithm with a throughput demand of  $W$  is at most that of the optimal schedule with a throughput demand  $2\Gamma \cdot W$ .*

**Corollary 2.** *If the demand is  $W = \min_j w_j$  then the energy induced by Algorithm 1 is optimal (compared to the optimal solution with the same demand).*

*Throughput Maximization with Energy Constraint.* We use the algorithm in the previous section as a subroutine and make a dichotomy search in the feasible domain of the total throughput. The formal algorithm and the proof of the following theorem are given in the following.

---

**Algorithm 2.** Maximizing throughput under the energy constraint

---

- 1: For a throughput demand  $W$ , denote  $E(W)$  the consumed energy due to Algorithm 1.
  - 2: Let  $\epsilon > 0$  be a constant.
  - 3: Initially, set  $W \leftarrow \min_j w_j$  and  $\overline{W} \leftarrow \sum_j w_j$  where the sum is taken over all jobs  $j$ .
  - 4: **if**  $E(W) > E$  **then**
  - 5:     **return** the total throughput is 0
  - 6: **end if**
  - 7: **while**  $E((1 + \epsilon)W) \leq E$  and  $(1 + \epsilon)W \leq \overline{W}$  **do**
  - 8:      $W \leftarrow (1 + \epsilon)W$ .
  - 9: **end while**
  - 10: **return** the schedule which is the solution of Algorithm 1 with throughput demand  $W$ .
- 

**Theorem 2.** *Given an energy budget  $E$ , there exists a polynomial time algorithm with respect to the size of input and  $1/\epsilon$ , and which is  $2(\Gamma + 1)(1 + \epsilon)$ -approximate in throughput for arbitrarily small  $\epsilon > 0$ .*

### 3 Exact Algorithms for Non-Preemptive Scheduling

In this section, we consider schedules without preemption with a fixed number  $m$  of identical machines. So the parameters of a job  $j$  are the same on every machine. Without loss of generality, we assume that all parameters of the problem such as release dates, deadlines and processing volumes of jobs are *integer*. We rename jobs in non-decreasing order of their deadlines, i.e.  $d_1 \leq d_2 \leq \dots \leq d_n$ . We denote by  $r_{\min} := \min_{1 \leq j \leq n} r_j$  the minimum release date. Define  $\Omega$  as the set of release dates and deadlines, i.e.,  $\Omega := \{r_j | j = 1, \dots, n\} \cup \{d_j | j = 1, \dots, n\}$ . Let  $J(k, a, b) := \{j | j \leq k \text{ and } a \leq r_j < b\}$  be the set of jobs among the  $k$  first ones w.r.t. the EDF (Earliest Deadline First) order, whose release dates are within  $a$  and  $b$ . We consider *time vectors*  $\mathbf{a} = (a_1, a_2, \dots, a_m) \in \mathbb{R}_+^m$  where each component  $a_i$  is a time associated to the machines  $i$  for  $1 \leq i \leq m$ . We say that  $\mathbf{a} \preceq \mathbf{b}$  if  $a_i \leq b_i$  for every  $1 \leq i \leq m$ . Moreover,  $\mathbf{a} \prec \mathbf{b}$  if  $\mathbf{a} \preceq \mathbf{b}$  and  $\mathbf{a} \neq \mathbf{b}$ . The relation  $\preceq$  is a partial order over the time vectors. Given a vector  $\mathbf{a}$ , we denote by  $a_{\min} := \min_{1 \leq i \leq m} a_i$ .

*Observations.* We give some simple observations on non-preemptive scheduling with the objective of maximizing throughput under the energy constraint. First, it is well known that due to the convexity of the power function  $P(z) := z^\alpha$ , each job runs at a constant speed during its whole execution in an optimal schedule. This follows from Jensen’s Inequality. Second, for a restricted version of the problem in which there is a single machine, jobs have the same processing volume and are agreeable, the problem is already  $\mathcal{NP}$ -hard. That is proved by a simple reduction from KNAPSACK.

In the following sections, we show pseudo-polynomial-time exact algorithms for particular settings: (1) setting with equal length jobs; and (2) setting with agreeable jobs.

#### 3.1 Equal Processing Volume, $p_j = p \ \forall j$

**Definition 1.** Let  $\Theta_{a,b} := \{a + \ell \cdot \frac{b-a}{k} \mid k = 1, \dots, n \text{ and } \ell = 0, \dots, k \text{ and } a \leq b\}$  and  $\Theta := \bigcup \{\Theta_{a,b} | a, b \in \Omega\}$ . Moreover, let  $\Lambda := \{\frac{\ell \cdot p}{b-a} \mid \ell = 1, \dots, n \text{ and } a, b \in \Omega \text{ and } a < b\}$ .

**Lemma 3.** *There exists an optimal schedule in which the starting time and completion time of each job belong to the set  $\Theta$ . Consequently, each job is processed at some speed which belongs to  $\Lambda$ .*

**Definition 2.** For  $0 \leq w \leq W$ , define  $E_k(\mathbf{a}, \mathbf{b}, w, e)$  as the minimum energy consumption of a non-preemptive (non-migration) schedule  $\mathcal{S}$  such that

- $S \subset J(k, a_{\min}, b_{\min})$  and  $\sum_{j \in S} w_j \geq w$  where  $S$  is the set of jobs scheduled in  $\mathcal{S}$ ,
- if  $j \in S$  is assigned to machine  $i$  then it is entirely processed in  $[a_i, b_i]$  for every  $1 \leq i \leq m$ ,

- $\mathbf{a} \preceq \mathbf{b}$ ,
- for some machine  $1 \leq h \leq m$ , it is idle during interval  $[a_h, e]$ ,
- for arbitrary machines  $1 \leq i \neq i' \leq m$ ,  $b_{i'}$  is at least the last starting time of a job in machine  $i$ .

Note that  $E_k(\mathbf{a}, \mathbf{b}, w, e) = \infty$  if no such schedule  $\mathcal{S}$  exists.

**Proposition 1.** *One has  $E_0(\mathbf{a}, \mathbf{b}, 0, e) = 0$ ,  $E_0(\mathbf{a}, \mathbf{b}, w, e) = +\infty \forall w \neq 0$*

$$E_k(\mathbf{a}, \mathbf{b}, w, e) = \min \left\{ \begin{array}{l} E_{k-1}(\mathbf{a}, \mathbf{b}, w, e) \\ \min_{\substack{\mathbf{u} \in \Theta^m \\ \mathbf{a} \preceq \mathbf{u} \prec \mathbf{b} \\ s \in \Lambda, 1 \leq h \leq m, \\ e' = u_h + \frac{p}{s}, \\ r_k \leq u_h < e' \leq d_k \\ 0 \leq w' \leq w - w_k}} \left\{ \begin{array}{l} E_{k-1}(\mathbf{a}, \mathbf{u}, w', e) + \frac{p^\alpha}{(e' - u_h)^{\alpha-1}} \\ + E_{k-1}(\mathbf{u}, \mathbf{b}, w - w' - w_k, e') \end{array} \right\} \end{array} \right.$$

**Theorem 3.** *The dynamic program in Proposition 1 has a running time of  $O(n^{12m+7}W^2)$ .*

### 3.2 Agreeable Jobs

In this section, we focus on another important family of instances. More precisely, we assume that the jobs have *agreeable* deadlines, i.e. for any pair of jobs  $i$  and  $j$ , one has  $r_i \leq r_j$  if and only if  $d_i \leq d_j$ .

By using a similar discretization as in the previous subsection, we can obtain the following result:

**Theorem 4.** *The weighted throughput problem for agreeable instances can be solved by dynamic programming in  $O(n^{2m+2}V^{2m+1}Wm)$  time.*

## References

1. Albers, S.: Energy-efficient algorithms. *Commun. ACM* **53**(5), 86–96 (2010)
2. Albers, S., Antoniadis, A., Greiner, G.: On multi-processor speed scaling with migration: extended abstract. In: *Proc. 23rd Annual ACM SPAA*, pp. 279–288. ACM (2011)
3. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: *Proc. 19th Annual ACM SPAA*, pp. 289–298. ACM (2007)
4. Angel, E., Bampis, E., Chau, V.: Throughput maximization in the speed-scaling setting. In: *STACS*, vol. 25. LIPIcs, pp. 53–62 (2014)
5. Angel, E., Bampis, E., Chau, V., Letsios, D.: Throughput Maximization for Speed-Scaling with Agreeable Deadlines. In: Chan, T.-H.H., Lau, L.C., Trevisan, L. (eds.) *TAMC 2013*. LNCS, vol. 7876, pp. 10–19. Springer, Heidelberg (2013)
6. Angel, E., Bampis, E., Kacem, F., Letsios, D.: Speed Scaling on Parallel Processors with Migration. In: Kaklamanis, C., Papatheodorou, T., Spirakis, P.G. (eds.) *Euro-Par 2012*. LNCS, vol. 7484, pp. 128–140. Springer, Heidelberg (2012)
7. Antoniadis, A., Huang, C.-C.: Non-preemptive speed scaling. *J. Scheduling* **16**(4), 385–394 (2013)

8. Antoniadis, A., Huang, C.-C., Ott, S., Verschae, J.: How to Pack Your Items When You Have to Buy Your Knapsack. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 62–73. Springer, Heidelberg (2013)
9. Bampis, E., Kononov, A., Letsios, D., Lucarelli, G., Nemparis, I.: From Preemptive to Non-preemptive Speed-Scaling Scheduling. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 134–146. Springer, Heidelberg (2013)
10. Bampis, E., Kononov, A., Letsios, D., Lucarelli, G., Sviridenko, M.: Energy efficient scheduling and routing via randomized rounding. In: FSTTCS, vol. 24. LIPIcs, pp. 449–460 (2013)
11. Bampis, E., Letsios, D., Lucarelli, G.: Green Scheduling, Flows and Matchings. In: Chao, K.-M., Hsu, T., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 106–115. Springer, Heidelberg (2012)
12. Bansal, N., Chan, H.-L., Lam, T.-W., Lee, L.-K.: Scheduling for Speed Bounded Processors. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 409–420. Springer, Heidelberg (2008)
13. Brucker, P.: Scheduling Algorithms, 5th edn. Springer Publishing Company, Incorporated (2010)
14. Carr, R.D., Fleischer, L., Leung, V.J., Phillips, C.A.: Strengthening integrality gaps for capacitated network design and covering problems. In: Proc. 11th ACM-SIAM SODA, pp. 106–115 (2000)
15. Chan, H.-L., Chan, W.-T., Lam, T. W., Lee, L.-K., Mak, K.-S., Wong, P.W. H.: Energy efficient online deadline scheduling. In: SODA, pp. 795–804. SIAM (2007)
16. Chan, H.-L., Lam, T.-W., Li, R.: Tradeoff between Energy and Throughput for Online Deadline Scheduling. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 59–70. Springer, Heidelberg (2011)
17. Cohen-Addad, V., Li, Z., Mathieu, C., Milis, I.: Energy-efficient algorithms for non-preemptive speed-scaling. Research report, [arXiv:1402.4111v2](https://arxiv.org/abs/1402.4111v2) [cs.DS] (2014)
18. Devanur, N.R., Jain, K.: Online matching with concave returns. In: Proc. 44th ACM STOC, pp. 137–144 (2012)
19. Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In: Proc. 21st Annual ACM SPAA, pp. 11–18. ACM (2009)
20. Huang, C.-C., Ott, S.: New Results for Non-Preemptive Speed Scaling. In: Csuhanj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 360–371. Springer, Heidelberg (2014)
21. Kling, P., Pietrzyk, P.: Profitable scheduling on multiple speed-scalable processors. In: Proc. 25th Annual ACM SPAA, pp. 251–260 (2013)
22. Lawler, E.L.: A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research* **26**(1), 125–133 (1990)
23. Li, M.: Approximation algorithms for variable voltage processors: Min energy, max throughput and online heuristics. *Theor. Comput. Sci.* **412**(32), 4074–4080 (2011)
24. Frances Yao, F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS, pp. 374–382. IEEE Computer Society (1995)

# Speed-Scaling with No Preemptions

Evripidis Bampis<sup>1</sup>, Dimitrios Letsios<sup>2</sup>, and Giorgio Lucarelli<sup>1,3</sup>(✉)

<sup>1</sup> Sorbonne Universités, UPMC Univ. Paris 06, UMR 7606, LIP6, Paris, France

Evripidis.Bampis@lip6.fr

<sup>2</sup> Institut für Informatik, Technische Universität München, Munich, Germany

letsios@in.tum.de

<sup>3</sup> Université Grenoble-Alpes, INP, UMR 5217, LIG, Grenoble, France

giorgio.lucarelli@inria.fr

**Abstract.** We revisit the non-preemptive speed-scaling problem, in which a set of jobs have to be executed on a single or a set of parallel speed-scalable processor(s) between their release dates and deadlines so that the energy consumption to be minimized. We adopt the speed-scaling mechanism first introduced in [Yao et al., FOCS 1995] according to which the power dissipated is a convex function of the processor's speed. Intuitively, the higher is the speed of a processor, the higher is the energy consumption. For the single-processor case, we improve the best known approximation algorithm by providing a  $(1 + \epsilon)^\alpha \tilde{B}_\alpha$ -approximation algorithm, where  $\tilde{B}_\alpha$  is a generalization of the Bell number. For the multiprocessor case, we present an approximation algorithm of ratio  $\tilde{B}_\alpha((1 + \epsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$  improving the best known result by a factor of  $(\frac{5}{2})^{\alpha-1}(\frac{w_{\max}}{w_{\min}})^\alpha$ . Notice that our result holds for the fully heterogeneous environment while the previous known result holds only in the more restricted case of parallel processors with identical power functions.

## 1 Introduction

Speed-scaling (or dynamic voltage scaling) is one of the main mechanisms to save energy in modern computing systems. According to this mechanism, the speed of each processor may dynamically change over time, while the energy consumed by the processor is proportional to a convex function of the speed. More precisely, if the speed of a processor is equal to  $s(t)$  at a time instant  $t$ , then the power dissipated is  $P(s(t)) = s(t)^\alpha$ , where  $\alpha > 1$  is a small constant. For example, the value of  $\alpha$  is theoretically between two and three for CMOS devices, while some experimental studies showed that  $\alpha$  is rather smaller: 1.11

---

E. BAMPIS, D. LETSIOS and G. LUCARELLI are partially supported by the project ALGONOW, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the program THALES and by the project Mathematical Programming and Non-linear Combinatorial Optimization under the program PGMO. D. LETSIOS is partially supported by the German Research Foundation, project AL-464/7-1. G. LUCARELLI is supported by the project Moebus funded by ANR.

for Intel PXA 270, 1.62 for Pentium M770 and 1.66 for a TCP offload engine [21]. Intuitively, higher speeds lead to higher energy consumption. The energy consumption is the integral of the power over time, i.e.,  $E = \int P(s(t))dt$ .

In order to handle the energy consumption in a computing system with respect to the speed-scaling mechanism, we consider the following scheduling problem. We are given a set of jobs and a single processor or a set of parallel processors. Each job is characterized by a release date, a deadline and an amount of workload that has to be executed between the job's release date and deadline. The objective is to find a feasible schedule that minimizes the energy consumption. In order to describe such a feasible schedule, we have to determine not only the job that has to be executed on every processor at each time instant, but also the speed of each processor.

Speed-scaling scheduling problems have been extensively studied in the literature. Since the seminal paper by Yao et al. [22] in 1995 until very recently, all the energy minimization works considered the *preemptive* case in which the execution of a job may be interrupted and restarted later on the same or even on a different processor (*migratory* case). However, the last three years, there are some works studying the *non-preemptive* case. In this paper, we improve the best known approximation algorithms for the non-preemptive case for both the single-processor and the multiprocessor environments.

*Problem definition and notation.* We consider a set  $\mathcal{J}$  of  $n$  jobs, each one characterized by an amount of work  $w_j$ , a release date  $r_j$  and a deadline  $d_j$ . We will consider both the single-processor and the multiprocessor cases. If the speed of a processor is equal to  $s(t)$  at a time instant  $t$ , then the power dissipated is  $P(s(t)) = s(t)^\alpha$ , where  $\alpha > 1$  is a small constant. In the multiprocessor environment, we denote by  $\mathcal{P}$  the set of the  $m$  available parallel processors. Moreover, we distinguish between the *homogeneous* and the *heterogeneous* multiprocessor cases. In the latter one, we assume that each processor  $i \in \mathcal{P}$  has a different constant  $\alpha_i$ , capturing in this way the existence of processors with different energy consumption rate. For simplicity, we define  $\alpha = \max_{i \in \mathcal{P}} \{\alpha_i\}$ . Moreover, in the *fully heterogeneous* case we additionally assume that each job  $j \in \mathcal{J}$  has a different work  $w_{i,j}$ , release date  $r_{i,j}$  and deadline  $d_{i,j}$  on each processor  $i \in \mathcal{P}$ . In all cases, the objective is to find a schedule that minimizes the energy consumption,  $E = \int P(s(t))dt$ , with respect to the speed-scaling mechanism, such that each job  $j \in \mathcal{J}$  is executed during its *life* interval  $[r_j, d_j]$ . The results presented in this paper assume that the preemption of jobs is not allowed; and hence neither their migration in the multiprocessor environments.

In what follows, we denote by  $w_{\max}$  and  $w_{\min}$  the maximum and the minimum work, respectively, among all jobs. Moreover, we call an instance *agreeable* if earlier released jobs have earlier deadlines, i.e., for each  $j$  and  $j'$  with  $r_j \leq r_{j'}$  then  $d_j \leq d_{j'}$ . Finally, given a schedule  $\mathcal{S}$  we denote by  $E(\mathcal{S})$  its energy consumption.

*Related work.* In [22], a polynomial-time algorithm has been presented that finds an optimal preemptive schedule when a single processor is available. In



the case where the preemption and also the migration of jobs are allowed, several polynomial-time algorithms have been proposed when a set of homogeneous parallel processors is available [3, 6, 10, 12], while in the fully heterogeneous environment an  $OPT + \epsilon$  algorithm with complexity polynomial to  $\frac{1}{\epsilon}$  has been presented in [9]. In the case where the preemption of jobs is allowed but not their migration, the problem becomes strongly NP-hard even if all jobs have equal release dates and equal deadlines [4]. For this special case, the authors in [4] observed that a PTAS can be derived from [17]. For arbitrary release dates and deadlines, a  $B_{\lceil\alpha\rceil}$ -approximation algorithm is known [16], where  $B_{\lceil\alpha\rceil}$  is the  $\lceil\alpha\rceil$ -th Bell number. This result has been extended in [9] for the fully heterogeneous environment, where an approximation algorithm of ratio  $(1 + \epsilon)^\alpha \tilde{B}_\alpha$  has been presented, where  $\tilde{B}_\alpha = \sum_{k=0}^{\infty} \frac{k^\alpha e^{-1}}{k!}$  is a generalization of the Bell number that is also valid for fractional values of  $\alpha$ .

When preemptions are not allowed, Antoniadis and Huang [7] proved that the single-processor case is strongly NP-hard, while they have also presented a  $2^{5\alpha-4}$ -approximation algorithm. In [9], an approximation algorithm of ratio  $2^{\alpha-1}(1 + \epsilon)^\alpha \tilde{B}_\alpha$  has been proposed, improving the ratio given in [7] for any  $\alpha < 114$ . Recently, an approximation algorithm of ratio  $(12(1 + \epsilon))^{\alpha-1}$  is given in [15], improving the approximation ratio for any  $\alpha > 25$ . Moreover, the relation between preemptive and non-preemptive schedules in the energy-minimization setting has been studied in [8]. The authors show that starting from the optimal preemptive solution created by the algorithm in [22], it is possible to obtain a non-preemptive solution which guarantees an approximation ratio of  $(1 + \frac{w_{\max}}{w_{\min}})^\alpha$ . In the special case where all jobs have equal work this leads to a constant factor approximation of  $2^\alpha$ . Recently, for this special case, Angel et al. [5] and Huang and Ott [18], independently, proposed an optimal polynomial-time algorithm based on dynamic programming. Note also that for agreeable instances the single-processor non-preemptive speed-scaling problem can be solved to optimality in polynomial time, as the algorithm proposed by Yao et al. [22] for the preemptive case returns a non-preemptive schedule for agreeable instances.

For homogeneous multiprocessors when preemptions are not allowed, an approximation algorithm with ratio  $m^\alpha (\frac{n}{\sqrt{n}})^{\alpha-1}$  has been presented in [8]. More recently, Cohen-Addad et al. [15] proposed an algorithm of ratio  $(\frac{5}{2})^{\alpha-1} \tilde{B}_\alpha ((1 + \epsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$ , transforming the problem to the fully heterogeneous preemptive non-migratory case and using the approximation algorithm proposed in [9]. This algorithm leads to an approximation ratio of  $2(1 + \epsilon)^\alpha 5^{\alpha-1} \tilde{B}_\alpha$  for the case where all jobs have equal work. The authors in [15] observe also that their algorithm can be used when each job  $j \in \mathcal{J}$  has a different work  $w_{i,j}$  on each processor  $i \in \mathcal{P}$ , by losing an additional factor of  $(\frac{w_{\max}}{w_{\min}})^\alpha$ .

Several other results concerning scheduling problems in the speed-scaling setting have been presented, involving the optimization of some Quality of Service (QoS) criterion under a budget of energy, or the optimization of a linear combination of the energy consumption and some QoS criterion (see for example [11, 13, 20]). Moreover, two other energy minimization variants of the speed-scaling model have been studied in the literature, namely the *bounded speed*

*model* in which the speeds of the processors are bounded above and below (see for example [14]), and the *discrete speed model* in which the speeds of the processors can be selected among a set of discrete speeds (see for example [19]). The interested reader can find more details in the surveys [1, 2].

*Our contribution.* In Section 2 we revisit the single-processor non-preemptive speed-scaling problem, and we present an approximation algorithm of ratio  $(1 + \varepsilon)^{\alpha-1} \tilde{B}_\alpha$  which becomes the best algorithm for any  $\alpha \leq 77$ . Recall that in practice  $\alpha$  is a small constant and usually  $\alpha \in (1, 3]$ . In [8], where the relation between preemptive and non-preemptive schedules has been explored, an example has been proposed which shows that the ratio of the energy consumption of the optimal non-preemptive schedule over the energy consumption of the optimal preemptive schedule can be  $\Omega(n^{\alpha-1})$ . A similar example was used in [15] to show that the standard configuration linear programming formulation has the same integrality gap. In both cases,  $w_{\max} = n$  and  $w_{\min} = 1$  and the worst-case ratio of the energy consumption of the optimal non-preemptive schedule over the energy consumption of the optimal preemptive one can be seen as  $\Omega((\frac{w_{\max}}{w_{\min}})^{\alpha-1})$ . In this direction, a  $(1 + \frac{w_{\max}}{w_{\min}})^{\alpha-1}$ -approximation algorithm for the single-processor case has been presented in [8]. To overcome the above lower bound, all known constant-factor approximation algorithms for the single-processor problem [7, 9, 15] consider an initial partition of the time horizon into some specific intervals defined by the so-called *landmarks*. These intervals are defined in such a way that there is not a job whose life interval is included in one of them. Intuitively, this partition is used in order to improve the lower bound by focusing on special preemptive schedules that can be transformed to a feasible non-preemptive schedule without loosing a lot in terms of approximation. Here, we are able to avoid the use of this partition improving in this way the result of [9] by a factor of  $2^{\alpha-1}$ . In order to do that, we modify the *configuration linear program* proposed in [9] by including an additional structural property that is valid for any feasible non-preemptive schedule. This property helps us to obtain a “good” preemptive schedule after a randomized rounding procedure. We transform this “good” preemptive schedule to a new instance of the energy-minimization single-processor problem that is *agreeable* by choosing in an appropriate way new release dates and deadlines for the jobs. In this way, it is then sufficient to apply the algorithm proposed in [22] in order to get a non-preemptive schedule of energy consumption at most the energy consumption of the preemptive one.

In Section 3 we consider the fully heterogeneous non-preemptive speed-scaling problem, and we improve the approximation ratio of  $(\frac{w_{\max}}{w_{\min}})^\alpha (\frac{5}{2})^{\alpha-1} \tilde{B}_\alpha ((1 + \varepsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$  given in [15] to  $\tilde{B}_\alpha ((1 + \varepsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$ . Consecutively, our result generalizes and improves the approximation ratio for the equal-works case from  $2(1 + \varepsilon)^\alpha 5^{\alpha-1} \tilde{B}_\alpha$  to  $(2(1 + \varepsilon))^\alpha \tilde{B}_\alpha$ . Note also that we generalize the machine environment and we pass from the homogeneous with different  $w_{i,j}$ 's to the fully heterogeneous one. Our algorithm combines two basic ingredients: the  $\tilde{B}_\alpha (1 + \varepsilon)^\alpha$ -approximation algorithm of [9] for the fully heterogeneous preemptive non-migratory speed-scaling problem and the  $(1 + \frac{w_{\max}}{w_{\min}})^\alpha$ -approximation

algorithm of [8] for the single-processor non-preemptive speed-scaling problem. The first algorithm is used in order to assign the jobs to the processors, while the second one to get a non-preemptive schedule for each processor independently. The key observation here is that the algorithm for the single-processor non-preemptive case presented in [8] transforms the optimal preemptive schedule obtained by the algorithm in [22] into a non-preemptive one. In this way, its approximation ratio is computed with respect to the energy consumption of the optimal preemptive schedule, which is a lower to the energy consumption of the optimal non-preemptive schedule.

We summarize our results with respect to the existing bibliography in Table 1.

**Table 1.** Comparison of the approximation ratios obtained in this paper with the previously best known approximation ratios

Machine environment	Previous known result	Our results
single-processor	$2^{\alpha-1}(1+\epsilon)^\alpha \tilde{B}_\alpha$ [9] $(12(1+\epsilon))^{\alpha-1}$ [15]	$(1+\epsilon)^\alpha \tilde{B}_\alpha$
homogeneous	$(\frac{5}{2})^{\alpha-1} \tilde{B}_\alpha ((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}}))^\alpha$ [15]	$\tilde{B}_\alpha ((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}}))^\alpha$
homogeneous with $w_{i,j}$ 's	$(\frac{5}{2})^{\alpha-1} \tilde{B}_\alpha ((1+\epsilon)(1+\frac{w_{\max}}{w_{\min}})\frac{w_{\max}}{w_{\min}})^\alpha$ [15]	
fully heterogeneous		

## 2 Single-Processor

In this section we consider the single-processor non-preemptive case and we present an approximation algorithm of ratio  $(1+\epsilon)\tilde{B}_\alpha$ , improving upon the previous known results [7,9,15] for any  $\alpha \leq 77$ . Our algorithm is based on a linear programming formulation combining ideas from [9,15] and the randomized rounding proposed in [9].

Before formulating the problem as a linear program we need to discretize the time into slots. Consider the set of all different release dates and deadlines of jobs in increasing order, i.e.,  $t_1 < t_2 < \dots < t_k$ . For each  $\ell, 1 \leq \ell \leq k-1$ , we split the time between  $t_\ell$  and  $t_{\ell+1}$  into  $n^2(1+\frac{1}{\epsilon})$  equal length slots as proposed in [18]. Let  $\mathcal{T}$  be the set of all created slots. Henceforth, we will consider only solutions in which each slot can be occupied by at most one job which uses the whole slot. Huang and Ott [18] proved that this can be done by losing a factor of  $(1+\epsilon)^{\alpha-1}$ .

Our formulation is based on the configuration linear program which was proposed in [9]. In [15], an additional constraint was used for the single-processor non-preemptive problem. This constraint implies that the life interval of a job cannot be included to the execution interval of another job. We explicitly incorporate this constraint in the definition of the set of configurations for each job. More specifically, for a job  $j \in \mathcal{J}$ , we define a configuration  $c$  to be a set of consecutive slots in  $[r_j, d_j]$  such that there is not another job  $j'$  whose life interval  $[r_{j'}, d_{j'}]$  is included in  $c$ . Let  $\mathcal{C}_j$  be the set of all possible configurations for

the job  $j$ . We introduce a binary variable  $x_{j,c}$  which is equal to one if the job  $j$  is executed according the configuration  $c \in \mathcal{C}_j$ . Let  $|c|$  be the length (in time) of the configuration  $c$ . Note that the number of configurations is polynomial as they only contain consecutive slots and the number of slots is also polynomial.

For notational convenience, we write  $t \in c$  if the slot  $t \in \mathcal{T}$  is part of the configuration  $c \in \mathcal{C}_j$  of job  $j \in \mathcal{J}$ . By the convexity of the power function, each job in an optimal schedule runs in a constant speed (see for example [22]). Hence, the quantity  $\frac{w_j^\alpha}{|c|^{\alpha-1}}$  corresponds to the energy consumed by  $j$  if it is executed according to  $c$ , as the constant speed that will be used for  $j$  is equal to  $\frac{w_j}{|c|}$ . Consider the following integer linear program.

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{J}} \sum_{c \in \mathcal{C}_j} x_{j,c} \frac{w_j^\alpha}{|c|^{\alpha-1}} \\ & \sum_{c \in \mathcal{C}_j} x_{j,c} \geq 1 \quad \forall j \in \mathcal{J} \\ & \sum_{j \in \mathcal{J}} \sum_{c \in \mathcal{C}_j: t \in c} x_{j,c} \leq 1 \quad \forall t \in \mathcal{T} \\ & x_{j,c} \in \{0, 1\} \quad \forall j \in \mathcal{J}, c \in \mathcal{C}_j \end{aligned}$$

The first constraint ensures that each job is executed according to a configuration. The second constraint implies that at each slot at most one configuration and hence at most one job can be executed.

We consider the randomized rounding procedure proposed in [9] for the fully heterogeneous preemptive non-migratory speed-scaling problem, adapted to the single processor environment. More specifically, for each job  $j \in \mathcal{J}$  we choose at random with probability  $x_{j,c}$  a configuration  $c \in \mathcal{C}_j$ . By doing this, more than one jobs may be assigned in a slot  $t \in \mathcal{T}$  which has as a result to get a non-feasible schedule. In order to deal with this infeasibility, for each slot  $t \in \mathcal{T}$  we perform an appropriate speed-up that leads to a feasible preemptive schedule. The above procedure is described formally in Algorithm 1.

---

**Algorithm 1**

---

- 1: Solve the configuration LP relaxation.
  - 2: For each job  $j \in \mathcal{J}$ , choose a configuration at random with probability  $x_{j,c}$ .
  - 3: Let  $w_j(t)$  be the amount of work executed for job  $j$  during the slot  $t \in \mathcal{T}$  according to its chosen configuration.
  - 4: Set the processor's speed during  $t$  as if  $\sum_{j \in \mathcal{J}} w_j(t)$  units of work are executed with constant speed during the entire  $t$ , i.e.,  $\sum_{j \in \mathcal{J}} w_j(t)/|t|$ , where  $|t|$  is the length of  $t$ .
  - 5: **return** the obtained schedule  $\mathcal{S}_{pr}$ .
- 

The analysis of the above procedure in [9] is done independently for each slot, while the speed-up performed leads to a loss of a factor of  $\bar{B}_\alpha$  to the

approximation ratio. We can use exactly the same analysis and get the same approximation guarantee for our problem, that is

$$E(\mathcal{S}_{pr}) \leq \tilde{B}_\alpha \cdot LP^* \tag{1}$$

where  $LP^*$  is the objective value of an optimal solution of the configuration LP relaxation.

In what follows, given the feasible preemptive schedule  $\mathcal{S}_{pr}$  obtained by Algorithm 1, we will create a feasible non-preemptive schedule  $\mathcal{S}_{npr}$  of energy consumption at most  $E(\mathcal{S}_{pr})$ . In fact, we first create a restricted agreeable instance  $\mathcal{I}'$  of our initial instance  $\mathcal{I}$  based on  $\mathcal{S}_{pr}$ . Then, we will apply the algorithm proposed by Yao et al. [22] that finds the optimal preemptive schedule on a single processor, which turns to be a non-preemptive schedule since the instance is agreeable. A formal description of our algorithm follows.

---

**Algorithm 2**

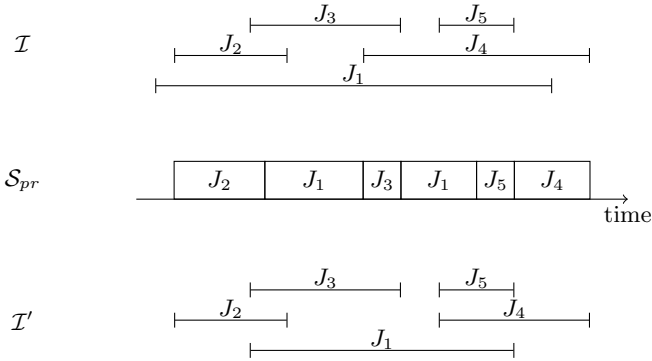
---

- 1: Run Algorithm 1 in the initial instance  $\mathcal{I}$  and get the schedule  $\mathcal{S}_{pr}$ .
  - 2: **for** each job  $j \in \mathcal{J}$  **do**
  - 3:   Let  $b_j$  be the time at which the first piece of  $j$  begins in  $\mathcal{S}_{pr}$ .
  - 4:   Let  $e_j$  be the time at which the last piece of  $j$  ends in  $\mathcal{S}_{pr}$ .
  - 5:   Select  $r'_j$  and  $d'_j$  such that
    - $r_j \leq r'_j \leq b_j$  and  $r'_j$  is minimum;
    - $e_j \leq d'_j \leq d_j$  and  $d'_j$  is maximum;
    - for any other job  $i \in \mathcal{J} \setminus \{j\}$ , it **cannot** hold that  $r'_j < r_i < d_i < d'_j$ .
  - 6: Create the instance  $\mathcal{I}'$  in which each job  $j \in \mathcal{J}$  has:
    - release date  $r'_j$ ,
    - deadline  $d'_j$ ,
    - work  $w_j$ .
  - 7: Run the algorithm proposed in [22] in the transformed instance  $\mathcal{I}'$  and get the schedule  $\mathcal{S}_{npr}$ .
  - 8: **return**  $\mathcal{S}_{npr}$ .
- 

An example of the above transformation is given in Fig. 1. In this picture, the life intervals of jobs  $J_1$  and  $J_4$  are shortened. For example, in the preemptive schedule  $\mathcal{S}_{pr}$  the job  $J_4$  is executed on the right of the job  $J_5$ . Hence, in the restricted instance we cut down the part of the life interval of  $J_4$  which is on the left of the release date of  $J_5$ . Intuitively, we decide if  $J_4$  should be executed on the left or on the right of  $J_5$  with respect to  $\mathcal{S}_{pr}$  and we transform the initial instance appropriately.

**Lemma 1.** *The restricted instance  $\mathcal{I}'$  is agreeable.*

*Proof.* Assume for contradiction that there are two jobs  $i, j \in \mathcal{J}$  in  $\mathcal{I}'$  such that  $r'_j < r'_i < d'_i < d'_j$ . The algorithm did not select a smaller  $r'_i$  because either there is a job  $k \in \mathcal{J}$  such that  $r'_i = r_k < d_k < d'_i$  or  $r'_i = r_i$ . In the first case, we have that  $r'_j < r_k < d_k < d'_j$ , which is a contradiction to the definition of



**Fig. 1.** The transformation of an instance  $\mathcal{I}$  into a restricted (agreeable) instance  $\mathcal{I}'$  based on the feasible preemptive schedule  $\mathcal{S}_{pr}$

configurations and the selection of  $r'_j$ . In the second case, the algorithm did not select a bigger  $d'_i$  because either there is a job  $\ell \in \mathcal{J}$  such that  $r'_i < r_\ell < d_\ell = d'_i$  or  $d'_i = d_i$ . In both last subcases we have again a contradiction, as either  $r'_j < r_\ell < d_\ell < d'_j$  or  $r'_j < r_i < d_i < d'_j$ .  $\square$

**Theorem 1.** *Algorithm 2 achieves an approximation ratio of  $(1 + \varepsilon)^{\alpha-1} \tilde{B}_\alpha$  for the single-processor non-preemptive speed-scaling problem.*

*Proof.* By construction, the life interval of each job  $j \in \mathcal{J}$  in the restricted instance  $\mathcal{I}'$  is a superset of its execution interval in  $\mathcal{S}_{pr}$ , i.e.,  $[b_j, e_j] \subseteq [r'_j, d'_j]$ . Hence, the schedule  $\mathcal{S}_{npr}$  is a feasible preemptive schedule for  $\mathcal{I}'$ .

By Lemma 1,  $\mathcal{I}'$  is an agreeable instance. Thus, by applying the algorithm proposed by Yao et al. [22], the schedule  $\mathcal{S}_{npr}$  is a non-preemptive schedule for  $\mathcal{I}'$ . Moreover, the life interval of each job  $j \in \mathcal{J}$  in  $\mathcal{I}'$  is a subset of its life interval in the initial instance  $\mathcal{I}$ , i.e.,  $[r'_j, d'_j] \subseteq [r_j, d_j]$ . Hence, the schedule  $\mathcal{S}_{npr}$  is a feasible non-preemptive schedule for  $\mathcal{I}$ .

Concerning the energy consumption, it holds that  $E(\mathcal{S}_{npr}) \leq E(\mathcal{S}_{pr})$  since  $\mathcal{S}_{npr}$  is an optimal schedule for  $\mathcal{I}'$  for both preemptive and non-preemptive versions. Hence, by using Equation (1) we have that  $E(\mathcal{S}_{npr}) \leq \tilde{B}_\alpha \cdot LP^*$ . Finally, taking into account the factor we loose by the discretization of the time proposed in [18], the theorem follows.  $\square$

### 3 Parallel Processors

In this section we consider the fully heterogeneous multiprocessor case and we propose an approximation algorithm of ratio  $\tilde{B}_\alpha \left( (1 + \epsilon) \left( 1 + \frac{w_{\max}}{w_{\min}} \right) \right)^\alpha$ , generalizing the recent result by Cohen-Addad et al. [15] from the homogeneous with different  $w_{i,j}$ 's to the fully heterogeneous environment and improving their ratio by a factor of  $\left( \frac{w_{\max}}{w_{\min}} \right)^\alpha \left( \frac{5}{2} \right)^{\alpha-1}$ . Our algorithm uses the following result proposed in [8].

**Theorem 2.** [8] *There is an approximation algorithm for the single-processor non-preemptive speed-scaling problem that returns a schedule  $\mathcal{S}$  with energy consumption  $E(\mathcal{S}) \leq (1 + \frac{w_{\max}}{w_{\min}})^\alpha E(\mathcal{S}_{pr}^*) \leq (1 + \frac{w_{\max}}{w_{\min}})^\alpha E(\mathcal{S}_{npr}^*)$ , where  $\mathcal{S}_{pr}^*$  and  $\mathcal{S}_{npr}^*$  are the optimal schedules for the preemptive and the non-preemptive case, respectively.*

The key observation in the above theorem concerns the first inequality of Theorem 2 that the energy consumption of the non-preemptive schedule  $\mathcal{S}$  created by the algorithm in [8] is bounded within a factor of  $(1 + \frac{w_{\max}}{w_{\min}})^\alpha$  by the energy consumption of the optimal preemptive schedule  $\mathcal{S}_{pr}^*$ . Based on this, we propose Algorithm 3 which uses the  $(1 + \epsilon)^\alpha \tilde{B}_\alpha$ -approximation algorithm proposed in [9] for the fully heterogeneous preemptive non-migratory speed-scaling problem to find a good assignment of the jobs to the processors and then applies Theorem 2 to create a non-preemptive schedule independently for each processor.

---

**Algorithm 3**

---

- 1: Find a preemptive non-migratory schedule  $\mathcal{S}$  using the algorithm proposed in [9] for the fully heterogeneous environment.
  - 2: **for** each processor  $i \in \mathcal{P}$  **do**
  - 3:   Let  $\mathcal{J}_i$  be the set of jobs assigned to processor  $i$  according to  $\mathcal{S}$ .
  - 4:   Find a single-processor non-preemptive schedule  $\mathcal{S}_{i,npr}$  using the algorithm proposed in [8] (Theorem 2) with input  $\mathcal{J}_i$ .
  - 5: **return** the non-preemptive schedule  $\mathcal{S}_{npr}$  which consists of the non-preemptive schedules  $\mathcal{S}_{i,npr}$ ,  $1 \leq i \leq m$ .
- 

**Theorem 3.** *Algorithm 3 achieves an approximation ratio of  $\tilde{B}_\alpha((1 + \epsilon)(1 + \frac{w_{\max}}{w_{\min}}))^\alpha$  for the fully heterogeneous non-preemptive speed-scaling problem.*

*Proof.* Consider first the schedule  $\mathcal{S}$  obtained in Line 1 of the algorithm, and let  $\mathcal{S}_{i,pr}$  be the (sub)schedule of  $\mathcal{S}$  that corresponds to the processor  $i \in \mathcal{P}$ . In other words, each  $\mathcal{S}_{i,pr}$  is a feasible preemptive schedule of the subset of jobs  $\mathcal{J}_i$ . As  $\mathcal{S}$  is a non-migratory schedule the subsets of jobs  $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_m$  are pairwise disjoint. Hence, we have that

$$\sum_{i \in \mathcal{P}} E(\mathcal{S}_{i,pr}) = E(\mathcal{S}) \leq (1 + \epsilon)^\alpha \tilde{B}_\alpha E(\mathcal{S}^*) \tag{2}$$

where  $\mathcal{S}^*$  is the optimal non-preemptive schedule for our problem and the inequality holds by the result in [9] and the fact that the energy consumption in an optimal preemptive-non-migratory schedule is a lower bound to the energy consumption of  $\mathcal{S}^*$ .

Consider now, for each processor  $i \in \mathcal{P}$ , the schedule  $\mathcal{S}_{i,npr}$  created in Line 4 of the algorithm. By Theorem 2 we have that

$$E(\mathcal{S}_{i,npr}) \leq \left(1 + \frac{w_{\max}}{w_{\min}}\right)^\alpha E(\mathcal{S}_{i,pr}^*)$$

where  $\mathcal{S}_{i,pr}^*$  is an optimal preemptive schedule for the subset of jobs  $\mathcal{J}_i$ . As  $\mathcal{S}_{i,pr}^*$  and  $\mathcal{S}_{i,pr}$  are schedules concerning the same set of jobs and  $\mathcal{S}_{i,pr}^*$  is the optimal preemptive schedule, we have that

$$E(\mathcal{S}_{i,npr}) \leq \left(1 + \frac{w_{\max}}{w_{\min}}\right)^\alpha E(\mathcal{S}_{i,pr}) \quad (3)$$

Since  $\mathcal{S}_{npr}$  is the concatenation of  $\mathcal{S}_{i,npr}$  for all  $i \in \mathcal{P}$ , and by using Equations (2) and (3), the theorem follows.  $\square$

Algorithm 3 can be also used for the case where all jobs have equal work on each processor, i.e., each job  $j \in \mathcal{J}$  has to execute an amount of work  $w_{i,j} = w_i$  if it is assigned on processor  $i \in \mathcal{P}$ . In this case we get the following result.

**Corollary 1.** *Algorithm 3 achieves a constant-approximation ratio of  $\tilde{B}_\alpha(2(1+\epsilon))^\alpha$  for the fully heterogeneous non-preemptive speed-scaling problem when all jobs have equal work on each processor.*

## 4 Conclusions

In this paper, we have presented algorithms with improved approximation ratios for both the single-processor and the multiprocessor environments. A challenging question left open in this work is the existence of a constant approximation ratio algorithm for the multiprocessor case. Also, there is a need for non-approximability results in the same vein as the one presented in [15].

## References

1. Albers, S.: Energy-efficient algorithms. *Communications of the ACM* **53**(5), 86–96 (2010)
2. Albers, S.: Algorithms for dynamic speed scaling. In: STACS. LIPIcs, vol. 9, pp. 1–11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
3. Albers, S., Antoniadis, A., Greiner, G.: On multi-processor speed scaling with migration: extended abstract. In: SPAA, pp. 279–288. ACM (2011)
4. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: SPAA, pp. 289–298. ACM (2007)
5. Angel, E., Bampis, E., Chau, V.: Throughput maximization in the speed-scaling setting. CoRR, abs/1309.1732 (2013)
6. Angel, Eric, Bampis, Evripidis, Kacem, Fadi, Letsios, Dimitrios: Speed Scaling on Parallel Processors with Migration. In: Kaklamanis, Christos, Papatheodorou, Theodore, Spirakis, Paul G. (eds.) Euro-Par 2012. LNCS, vol. 7484, pp. 128–140. Springer, Heidelberg (2012)
7. Antoniadis, Antonios, Huang, Chien-Chung: Non-preemptive Speed Scaling. In: Fomin, Fedor V., Kaski, Petteri (eds.) SWAT 2012. LNCS, vol. 7357, pp. 249–260. Springer, Heidelberg (2012)
8. Bampis, Evripidis, Kononov, Alexander, Letsios, Dimitrios, Lucarelli, Giorgio, Nemparis, Ioannis: From Preemptive to Non-preemptive Speed-Scaling Scheduling. In: Du, Ding-Zhu, Zhang, Guochuan (eds.) COCOON 2013. LNCS, vol. 7936, pp. 134–146. Springer, Heidelberg (2013)



9. Bampis, E., Kononov, A., Letsios, D., Lucarelli, G., Sviridenko, M.: Energy efficient scheduling and routing via randomized rounding. In: FSTTCS. LIPIcs, vol. 24, pp. 449–460. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
10. Bampis, Evripidis, Letsios, Dimitrios, Lucarelli, Giorgio: Green Scheduling, Flows and Matchings. In: Chao, Kun-Mao, Hsu, Tsan-sheng, Lee, Der-Tsai (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 106–115. Springer, Heidelberg (2012)
11. Bampis, Evripidis, Letsios, Dimitrios, Milis, Ioannis, Zois, Georgios: Speed Scaling for Maximum Lateness. In: Gudmundsson, Joachim, Mestre, Julián, Viglas, Taso (eds.) COCOON 2012. LNCS, vol. 7434, pp. 25–36. Springer, Heidelberg (2012)
12. Bingham, B.D., Greenstreet, M.R.: Energy optimal scheduling on multiprocessors with migration. In: ISPA, pp. 153–161. IEEE (2008)
13. Bunde, D.P.: Power-aware scheduling for makespan and flow. In: SPAA, pp. 190–196. ACM (2006)
14. Chan, H.-L., Chan, W.-T., Lam, T. W., Lee, L.-K., Mak, K.-S., Wong, P. W. H.: Energy efficient online deadline scheduling. In: SODA, pp. 795–804 (2007)
15. Cohen-Addad, V., Li, Z., Mathieu, C., Milis, I.: Energy-efficient algorithms for non-preemptive speed-scaling. In: WAOA. LNCS. Springer (2014)
16. Greiner, G., Nonner, T., Souza, A.: The bell is ringing in speed-scaled multiprocessor scheduling. In: SPAA, pp. 11–18. ACM (2009)
17. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM* **34**, 144–162 (1987)
18. Huang, Chien-Chung, Ott, Sebastian: New Results for Non-Preemptive Speed Scaling. In: Csuhaj-Varjú, Erzsébet, Dietzfelbinger, Martin, Ésik, Zoltán (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 360–371. Springer, Heidelberg (2014)
19. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. *SIAM Journal on Computing* **35**, 658–671 (2006)
20. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. *Theory of Computing Systems* **43**, 67–80 (2008)
21. Wierman, A., Andrew, L.L.H., Tang, A.: Power-aware speed scaling in processor sharing systems. In: INFOCOM, pp. 2007–2015. IEEE (2009)
22. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: FOCS, pp. 374–382. IEEE Computer Society (1995)

# **Computational Complexity**

# A Short Implicant of a CNF Formula with Many Satisfying Assignments

Daniel M. Kane<sup>1</sup> and Osamu Watanabe<sup>2(✉)</sup>

<sup>1</sup> Department of Computer Science/Department of Mathematics,  
University of California, San Diego, 9500 Gilman Drive #0404,  
La Jolla, CA 92093, USA

`dankane@math.stanford.edu`

<sup>2</sup> Department of Mathematical and Computing Sciences,  
Tokyo Institute of Technology, Tokyo, Japan

`watanabe@is.titech.ac.jp`

**Abstract.** Consider any Boolean function  $F(X_1, \dots, X_N)$  that has more than  $2^{-N^\delta} \cdot 2^N$  satisfying assignments for some  $\delta$ ,  $0 < \delta < 1$ , and that can be expressed by a CNF formula with at most  $N^d$  clauses for some  $d > 0$ . Then how many variables do we need to fix in order to satisfy  $F$ ? We show that one can always find some “short” partial assignment on which  $F$  evaluates to 1 by fixing at most  $\alpha N$  variables for some constant  $\alpha < 1$ ; that is,  $F$  has an implicant of size  $\leq \alpha N$ . A lower bound for such  $\alpha$  is also shown in terms of  $\delta$  and  $d$ . We also discuss an algorithm for obtaining a short partial assignment. For any  $\delta$  and  $\varepsilon$  such that  $0 < \delta + \varepsilon < 1$ , we show a deterministic algorithm that finds a short partial assignment in  $\tilde{O}(2^{N^\beta})$ -time<sup>1</sup> for some  $\beta < 1$  for any CNF formula with at most  $N^{1+\varepsilon}$  clauses having more than  $2^{-N^\delta} \cdot 2^N$  satisfying assignments. (This is an extended abstract, and some detailed explanations are omitted; see [6] for the details.)

## 1 Introduction

Consider any CNF formula Boolean function  $F(X_1, \dots, X_N)$  that has a relatively large number of satisfying assignments. Can we find some large subset of these satisfying assignments sharing some common partial assignment? We show that  $F$  has a relatively short implicant; that is, it is satisfied by some large set of satisfying assignments expressed by a short partial assignment.

To state our result we introduce some notation. Throughout this paper, let  $F$  be a given Boolean function over  $N$  variables, and we assume that it is given as a CNF formula with  $M$  clauses and that it has  $P2^N$  satisfying assignments, where  $P$  will be referred as the *sat. assignment ratio* of  $F$ . We introduce two parameters  $\delta$ ,  $0 < \delta < 1$ , and  $d > 0$ , and consider the following situation: (i)  $P \geq 2^{-N^\delta}$ , and (ii)  $M \leq N^d$ . For such a CNF formula  $F$ , we discuss the size of its implicant in terms of  $\delta$  and  $d$ . As our main result, we show that if  $\delta < 1$ , then

<sup>1</sup> By  $\tilde{O}(2^{N^\beta})$  we mean  $O(2^{N^\beta} \cdot N^{O(1)})$ .

one can always find some “short” and “satisfying” partial assignment, where by “short” we mean that it fixes  $\alpha N$  variables for some constant  $\alpha < 1$  and by “satisfying (or, sat.) partial assignment” we mean that  $F$  is evaluated to 1 (i.e., true) under this partial assignment. (In this paper, for any partial assignment, by its “size” we will mean the number of variables fixed by this assignment.)

If a Boolean function  $F$  has a short sat. partial assignment, then it has many sat. assignments. Our result shows that a certain converse relation holds provided that  $F$  is expressed as a CNF formula consisting of some fixed polynomial number of clauses. Our investigation is motivated by the hardness of the CNF-SAT problem, i.e., the satisfiability problem of general<sup>2</sup> CNF formulas. While the  $k$ -CNF-SAT problem for any constant  $k$  is solved by some  $\tilde{O}(c^n)$ -time algorithm for some  $c < 2$ , it has been open whether CNF-SAT has a similar nontrivial exponential-time algorithm. We would like to shed light to this open problem by considering the structure of sat. solutions of CNF-SAT. In fact, a quite strong result has been known for  $k$ -CNF formulas; Hirsch [5] showed that any  $k$ -CNF formula with sat. assignment ratio  $P$  has a partial assignment of size  $O(2^k \log(1/P))$ , which is sublinear in  $N$  when  $k$  is constant and  $P \geq 2^{-N^\delta}$ . Unfortunately, though, his argument does not seem to work for general CNF formulas. In fact, Hirsch proved the existence of a general CNF formula that has no sublinear size sat. partial assignment even though it has a large sat. assignment ratio, say,  $P \geq 0.5$ . We show here that even in the general case, it still has a  $(1 - \Omega(1))N$ -size sat. partial assignment. We hope that this structural property would be of some help for designing algorithms for CNF-SAT.

The Switching Lemma of Håstad [4] also can be used to discuss the existence of somewhat short satisfying partial assignments. For example, it is not so hard to show that any CNF formulas with some fixed polynomial number of clauses and constant sat. assignment ratio has a satisfying partial assignment of size  $\leq (1 - 1/\log N)N$ . But it seems that there is no trivial way to improve this bound. The contribution of this paper is to improve the upper bound to  $(1 - \Omega(1))N$  (even for much smaller sat. assignment ratio).

We also consider an algorithmic way to get such a short sat. partial assignment, and obtain a deterministic subexponential-time algorithm that finds one of short sat. partial assignments for CNF formulas with subquadratic number of clauses. More precisely, for any  $\delta$  and  $\varepsilon$  such that  $\delta + \varepsilon < 1$ , we can define a deterministic algorithm that takes any  $F$  satisfying (i) and (ii) with  $\delta$  and  $d = 1 + \varepsilon$  and computes its sat. partial assignment of size  $\leq \alpha N$  in  $\tilde{O}(2^{N^\beta})$ -time for some constants  $\alpha < 1$  and  $\beta < 1$ . Clearly, our deterministic algorithm for computing a short sat. partial assignment can be used for solving the CNF-SAT problem, and it has some advantages over previously known algorithms. An obvious randomized algorithm for the SAT problem for instances with many sat. assignments is to search for a sat. assignment by generating assignments uniformly at random. Such an algorithm finds a sat. assignment with probability  $\geq 2^{-N^\delta}$  for any function with sat. assignment ratio  $\geq 2^{-N^\delta}$ . Then for

<sup>2</sup> A CNF formula is called  $k$ -CNF if its all clauses consist of at most  $k$  literals. By a “general” CNF formula we mean a formula with no such restriction.

the CNF-SAT problem, we may design a deterministic algorithm by applying some good pseudo random sequence generator (*prg* in short) against CNF formulas to this randomized algorithm. That is, an algorithm that tries to find a sat. assignment among assignments generated by such a prg from all possible seeds. In order to ensure that this algorithm obtains some sat. assignment for any CNF formula with sat. assignment ratio  $\geq 2^{-N^\delta}$ , we need to choose the seed length of the prg so that a generated pseudo random sequence (of length  $N$ ) is  $\gamma := O(2^{-N^\delta})$  close to the uniform distribution for any CNF formula (with, say,  $N^{O(1)}$  clauses). For this application, the current best upper bound for the seed length is  $\tilde{O}(\log(1/\gamma)^2)$  (ignoring minor factors for our discussion) due to the prg proposed by De et al. [3]. For this seed length, the running time of the simple deterministic algorithm becomes  $\tilde{O}(2^{N^{2\delta}})$ , which is subexponential if  $\delta < 1/2$ . On the other hand, our algorithm's time bound  $\tilde{O}(2^{N^\beta})$  with  $\beta = 1 - \frac{(1-(\delta+\varepsilon))^2}{3}$  is subexponential if  $\delta + \varepsilon < 1$ .

## 2 Notation and Results

Throughout this paper, we will fix the usage of the following symbols: Let  $F$  be any Boolean function over  $N$  Boolean variables  $X_1, \dots, X_N$ , where  $N$  is our size parameter. We assume that  $F$  has  $P2^N$  sat. assignments where  $P \geq 2^{-N^\delta}$ , and that  $F$  is given as a CNF formula with  $M \leq N^d$  clauses for some  $d > 0$ . In this paper we regard parameters  $\delta$  and  $d$  as constants; whenever necessary, we may assume that  $N$  is large enough for each choice of  $\delta$  and  $d$ . We use  $|F|$  to denote the number of clauses in  $F$ , and for any clause  $C$ , we use  $|C|$  to denote the number of literals in  $C$ . The number of elements in a set  $W$  is denoted as  $\|W\|$ . Symbols  $\rho$  and  $\sigma$  are used to denote partial assignments over  $X_1, \dots, X_N$ . Any partial assignment  $\rho$  takes value 0, 1, or  $X_i$  on each variable  $X_i$ . We say that  $\rho$  *fixes* (the value of)  $X_i$  if  $\rho(X_i) = 0$  or 1, and that  $\rho$  *leaves  $X_i$  unassigned* if  $\rho(X_i) = X_i$ . By  $F|\rho$ , we mean a function evaluated by replacing each occurrence of  $X_i$  with  $\rho(X_i)$ . We say that  $\rho$  is a *sat. partial assignment* if  $F|\rho = 1$ ; this is a natural generalization of the standard satisfying assignment notion. We use  $\text{Fix}(\rho)$  to denote the set of variables that are fixed by  $\rho$ . Let  $\text{sat}(F)$  denote the set of sat. assignments of  $F$ . Then the *sat. assignment ratio of  $F$*  (denoted as  $\text{sat.ratio}(F)$ ) is defined by  $\text{sat.ratio}(F) = \|\text{sat}(F)\|/2^N$ . This quantity is naturally generalized to  $F|\rho$  for any partial assignment  $\rho$ , which is denoted as  $\text{sat.ratio}(F|\rho)$ . For any partial assignments  $\rho_1$  and  $\rho_2$  such that  $\text{Fix}(\rho_1) \cap \text{Fix}(\rho_2) = \emptyset$ , we write  $\rho_1 \circ \rho_2$  to denote a partial assignment fixing the coordinates fixed by  $\rho_1$  or  $\rho_2$  to the appropriate value and leaves others unassigned.

In this paper, we use symbols  $\alpha$  and  $\beta$  for some constants w.r.t.  $N$ , which are defined in terms of  $\delta$  and  $d$  (and some other technical parameters). On the other hand, symbol  $c$  is used to denote some constants independent from  $N$ ,  $\delta$ , and  $d$ . For simplifying our notation during the analysis, we will use some concrete constants such as 0.1, 0.5, etc. whenever we can choose them appropriately when  $N$  is sufficiently large. Also we will sometimes use three digit numbers, e.g., 0.99 to denote  $1 - o(1)$ . When stating our results formally, these constants and

numbers will be replaced with some corresponding standard notation such as  $O(N)$ ,  $\Omega(N)$ ,  $1 - o(1)$ , etc. We simply write  $\log$  for  $\log_2$  and  $\ln$  for  $\log_e$ . Let  $c_e = \log_2 e$ . When necessary, we write  $e^x$  and  $2^x$  as  $\exp(x)$  and  $\exp_2(x)$  respectively for showing the exponent clearly.

Our main result is now stated as follows.

**Theorem 1.** *For any  $\delta$ ,  $0 < \delta < 1$ , and for any  $d > 0$ , let  $F$  be any CNF formula such that (i) it has sat. assignment ratio  $P \geq \exp_2(-N^\delta)$ , and (ii) it consists of  $M \leq N^d$  clauses. Then it has some sat. partial assignment  $\hat{\rho}$  of size  $\leq \alpha N$ , where  $\alpha$  is defined by*

$$\alpha = 1 - \frac{1 - \delta}{cd} \exp_2 \left( -\frac{(1 + o(1))d}{1 - \delta} \right), \tag{1}$$

with some constant  $c > 0$ .

On the other hand, we show the following lower bound, which is much stronger than the one given in [5].

**Theorem 2.** *For any  $\delta$ ,  $0 < \delta < 1$ , and for any  $d \geq 1$ , consider  $\alpha$  defined by*

$$\alpha = \frac{d - (1 + o(1))}{d - \delta} > 1 - \frac{1 - \delta + o(1)}{d}. \tag{2}$$

Then we have some CNF formula  $F$  such that (i) it has sat. assignment ratio  $P \geq \exp_2(-N^\delta)$ , (ii) it consists of  $M \leq N^d$  clauses, and (iii) it has no sat. partial assignment  $\rho$  of size  $\leq \alpha N$ .

We also have an algorithmic version of Theorem 1 when the number of clauses is bounded by  $N^{1+\varepsilon}$  for some  $\varepsilon < 1$  such that  $\delta + \varepsilon < 1$  holds.

**Theorem 3.** *For any  $\delta > 0$  and  $\varepsilon > 0$  such that  $\delta + \varepsilon < 1$ , there exists a deterministic algorithm such that for any given CNF formula  $F$  satisfying (i) and (ii) of Theorem 1 w.r.t.  $\delta$  and  $d = 1 + \varepsilon$ , it runs in  $\tilde{O}(2^{N^\beta})$ -time for some  $\beta < 1$  and yields some sat. partial assignment  $\hat{\rho}$  for  $F$  of size  $\leq \alpha N$ , where  $\alpha$  is defined by*

$$\alpha = 1 - \frac{1 - (\delta + \varepsilon)}{c_1} \cdot \exp_2 \left( -\frac{c_2}{(1 - (\delta + \varepsilon))(1 - \delta)} \right) \tag{3}$$

with some constants  $c_1, c_2 \geq 1$ .

**Remark.** *We can show that the above time bound holds for any  $\beta$  such that  $\beta \geq 1 - \frac{(1 - (\delta + \varepsilon))^2}{3}$ .*

We recall some common bounds that will be used often in this paper. For any integer  $n \geq 1$ , we have

$$\left(1 - \frac{1}{n}\right)^n \leq e^{-1} \leq \left(1 - \frac{1}{n+1}\right)^n, \text{ and } \left(1 + \frac{1}{n}\right)^n \leq e \leq \left(1 + \frac{1}{n}\right)^{n+1}.$$

### 3 Upper Bound Proof

In this section we give a proof of Theorem 1, showing an upper bound on the size of short sat. partial assignments. Throughout this section, for any  $\delta$ ,  $0 < \delta < 1$  and any  $d > 0$ , we consider sufficiently large  $N$  and fix any  $F$  satisfying (i) and (ii) of the theorem w.r.t.  $\delta$  and  $d$ .

The key tool of our proof is the following lemma, which can be shown as a corollary of the analysis given by Hirsch [5]. By the *width* of a clause, we mean the number of literals appearing in the clause.

**Lemma 1.** *Consider any CNF formula consisting of clauses of width  $\leq k$  with sat. assignment ratio  $Q > 0$ . Then it has a partial satisfying assignment of size  $\leq 4c_e 2^k \log Q^{-1}$ .*

**Remark.** *We may consider  $k$  as a function in  $N$ . For simplicity, we assume that and  $Q < 1/4$ .*

Let us first see the outline of our proof. For given  $F$ , we show the existence of some partial assignment  $\rho_{12}$  that assigns some  $(1 - \eta)N$  variables for some  $\eta < 1$  and converts  $F$  to a formula consisting of *narrow clauses*, clauses of width, say,  $\leq 0.99(1 - \delta) \log N$ , while keeping relatively large sat. assignment ratio  $Q \geq 2^{-2N^\delta}$ . Then the theorem follows from the above lemma. For showing  $\rho_{12}$ , we use the idea of Ajtai introduced in [1] and define  $\rho_{12}$  in two stages. In the first stage, we define a partial assignment  $\rho_1$  to eliminate all *wide clauses*, clauses of width  $\geq Ad \ln N$  where  $A \geq 1$  is some parameter defined later. We then define  $\rho_2$  in the second stage that converts all clauses to narrow ones. We show that  $\rho_{12} = \rho_2 \circ \rho_1$  has the desired properties.

Now we explain each stage precisely from the first stage for defining  $\rho_1$ . We show a procedure for defining a sequence of partial assignments  $\sigma_1, \sigma_2, \dots, \sigma_T$  so that  $\rho_1$  is defined by  $\rho_1 = \sigma_T \circ \dots \circ \sigma_1$ . Intuitively, the main objective of the procedure is to eliminate wide clauses. Consider the situation where we have determined  $\sigma_1, \dots, \sigma_{t-1}$ , and let  $F_{t-1}$  denote  $F | \sigma_{t-1} \circ \dots \circ \sigma_1$ . Also let  $W$  denote the set of wide clauses in  $F_{t-1}$ . Note that there must be some variable  $X_i$  that appears in more than  $\frac{\|W\| Ad \log N}{N}$  clauses of  $W$ ; then either  $X_i$  or  $\bar{X}_i$  is a literal that appears more than  $\frac{\|W\| Ad \log N}{2N}$  clauses of  $W$ , which we call a *popular literal* among  $W$ . We would like to define  $\sigma_t$  to assign positively to one of such popular literals, thereby killing many wide clauses. But we should be careful not to reduce the sat. assignment ratio too much by this assignment. Here we check whether the assignment reduces the sat. assignment ratio too much, specifically, less than multiplying  $1 - p_1$ , and if so, use the opposite assignment to the popular literal. Note that this opposite assignment increases the sat. assignment ratio by  $1 + p_1$ . From this, we can show that such opposite assignments do not occur so many times (since the sat. assignment ratio cannot go beyond 1). Though very natural, this is a somewhat new technical point for implementing the idea of Ajtai for our problem.

Define the probability parameter  $p_1$  by  $p_1 = AN^{-(1-\delta)}/2$ , and using this  $p_1$ , we formally describe our idea as a procedure in Figure 1. We iterate this procedure until no wide clause exists.

```

procedure for  $\sigma_t$  (where  $t \geq 1$ )
// assume that  $\sigma_1, \dots, \sigma_{t-1}$  have been defined, and let  $F_{t-1}$  denote  $F|\sigma_{t-1} \circ \dots \circ \sigma_1$ .
  if  $F_{t-1}$  has no wide clause
    then stop and output the obtained sequence as  $\sigma_1, \dots, \sigma_T$ ;
   $W$  = the set of wide clauses in  $F_{t-1}$ ;
   $Y_i$  = (any) one of the popular literal (either  $X_i$  or  $\overline{X_i}$ ) among  $W$ ;
  if  $\text{sat.ratio}(F_{t-1}|(Y_i := 1)) \leq (1 - p_1) \cdot \text{sat.ratio}(F_{t-1})$ 
    then  $\sigma_t = (Y_i := 0)$ ; (Case I)
    else  $\sigma_t = (Y_i := 1)$ ; (Case II)
  //  $\sigma_t$  leaves the other variables unassigned.
    
```

**Fig. 1.** Procedure for defining  $\sigma_t$

**Lemma 2.** *Define  $T$  be the number of iterations of the above procedure needed until no wide clause exists in  $F_T$ . Then we have  $T \leq 6N/A$ . Also we have  $\text{sat.ratio}(F_T) \geq 0.99 \cdot 2^{-2N^\delta}$ .*

*Proof.* Let  $T_1$  and  $T_2$  denote respectively the number of iterations such that Case I and Case II occurs. We show that each of them is bounded by  $O(N/A)$ .

We first show that  $T_2 \leq 2N/A$ . For any  $t \geq 1$ , suppose that Case II occurs at the  $t$ th iteration of our procedure. That is, the algorithm finds some literal  $Y_i$  (either  $X_i$  or  $\overline{X_i}$ ) that is popular among the set  $W$  of wide clauses in  $F_{t-1}$ , and it indeed assigns true to  $Y_i$ . This assignment satisfies (and hence removes) more than  $\frac{\|W\|Ad \log N}{2N}$  clauses of  $W$ , which reduces the number of wide clauses by  $(1 - \frac{Ad \log N}{2N})$ . Thus, since we have initially at most  $M (\leq N^d)$  wide clauses, if Case II occurs for  $T'$  times, then the remaining number of wide clauses becomes at most

$$M \left( 1 - \frac{Ad \log N}{2N} \right)^{T'} < N^d \exp \left( -\frac{Ad \log N}{2N} T' \right) = N^d N^{-d \cdot \frac{A}{2N} \cdot T'}$$

Thus, the remaining number of wide clauses becomes less than 1 (that is, 0) if  $T' \geq 2N/A$ . Hence, Case II does not occur more than  $2N/A$  times, that is,  $T_2 \leq 2N/A$ .

We can also show here that the sat. ratio does not decrease so much by an assignment defined by this first stage. Note that the sat. ratio may decrease only by assignments defined at Case II. On the other hand, for any iteration  $t$  where  $Y_i$  is selected as a popular literal, Case II would not be chosen if the sat. ratio is increased by  $1 + p_1$  by an assignment  $Y_i := 0$ . Hence, when Case II is chosen, it is guaranteed that the sat. ratio does not get decreased less than  $1 - p_1$  by assigning  $Y_i$  true. Thus, from our bound for  $T_2$ , for any  $t$ th iteration of the procedure, we have  $\text{sat.ratio}(F|\sigma_t \circ \dots \circ \sigma_1) \geq \text{sat.ratio}(F)(1 - p_1)^{T_2}$ , which can be bounded by, say,  $0.99 \cdot 2^{-2N^\delta}$  by our choice of  $p_1$ . In particular, this bound holds when the iteration stops with no wide clause.

Next we give a bound  $T_1 \leq 4N/A$ . From the above, we know that the sat. ratio cannot be smaller than  $0.99 \cdot 2^{-2N^\delta}$  by the assignments of Case II. On the



other hand, at each step where Case I is chosen, the sat. ratio gets increased by  $1 + p_1$ . Hence, if Case I occurs  $T'$  times by some  $t$ th iteration, then the sat. ratio of  $F_t$  becomes at least  $0.99 \cdot 2^{-2N^\delta} (1 + p_1)^{T'}$ , which can be bounded by

$$0.99 \cdot 2^{-2N^\delta} \left( 1 + \frac{c_e A}{2} N^{-(1-\delta)} \right)^{T'} \geq 0.99^2 \exp_2 \left( -2N^\delta + 2N^\delta \frac{AT'}{4N} \right).$$

Thus, if  $T' > 4N/A$ , then the sat. ratio of  $F_t$  becomes larger than 1, a contradiction. Therefore we have  $T_1 \leq 4N/A$ . From these bounds the lemma follows.  $\square$

With  $A = 12$  use our procedure to define  $\rho_1 = \sigma_T \circ \dots \circ \sigma_1$ . Then Lemma 2 guarantees that  $F_T = F|\rho_1$  has no wide clause, it has sat. ratio  $\geq 0.99 \cdot 2^{-2N^\delta}$ , and  $\rho_1$  fixes at most  $6N/A = N/2$  variables. Next we consider the second stage to define  $\rho_2$  for converting all clauses of  $F_T$  to narrow ones. Without loss of generality (by renaming variable indices) we may assume that, for some  $N' \geq N/2$ ,  $\mathbf{X} = \{X_1, \dots, X_{N'}\}$  is the set of variables of  $F_T$ ; that is,  $X_1, \dots, X_{N'}$  are variables unassigned by  $\rho_1$ . The idea is to show the existence of some subset  $\mathbf{S}$  of  $\mathbf{X}$  such that (i) each clause of  $F_T$  has at most  $k = 0.99(1 - \delta) \log N$  variables in  $\mathbf{S}$ , and (ii)  $\|\mathbf{S}\| = \Omega(N)$ . Then from (i) it follows that any assignment to  $\mathbf{X} \setminus \mathbf{S}$  transform  $F_T$  to a formula consisting of only narrow clauses. From such partial assignments, we choose one with the largest sat. ratio as  $\rho_2$ .

**Lemma 3.** *Use notation as above. There exists a subset  $\mathbf{S}$  of  $\mathbf{X}$  (= the set of all variables of  $F_T$ ) such that (i) every clause in  $F_T$  has at most  $k = 0.99(1 - \delta) \log N$  variables in  $\mathbf{S}$ , and (ii)  $\|\mathbf{S}\| \geq 0.99\eta_d N/2$ , where*

$$\eta_d = \frac{0.99(1 - \delta)}{70d} \exp_2 \left( -\frac{d}{0.98(1 - \delta)} \right).$$

Hence,  $F_T|\rho'$  has only narrow clauses for any partial assignment  $\rho'$  that fixes all and only variables in  $\mathbf{X} \setminus \mathbf{S}$ . Furthermore, among such partial assignments, there exists some  $\rho_2$  such that  $\text{sat.ratio}(F_T|\rho_2) \geq 0.99 \cdot 2^{-2N^\delta}$  holds.

*Proof.* We generate  $\mathbf{S}$  randomly by selecting each  $X_i \in \mathbf{X}$  with probability  $\eta_d$  independently. Then with high probability, we have  $\|\mathbf{S}\| \geq 0.99\eta_d N'$  ( $\geq 0.99\eta_d N/2$ ) by Chernoff bound, we can bound the probability that  $\|\mathbf{S}\| < 0.99\eta_d N'$  occurs by, say, 0.1 (for sufficiently large  $N$ ).

Consider any clause  $C$  of  $F_T$ , and we estimate the probability that it has at least  $k = 0.99(1 - \delta) \log N$  literals in  $\mathbf{S}$ . For any fixed  $k$  literals in  $C$ , the probability that they (i.e., these variables) all selected in  $\mathbf{S}$  is  $\eta_d^k$ . Hence, by using the union bound, the probability that some  $k$  literals are all selected in  $\mathbf{S}$  is at most

$$\begin{aligned} \binom{Ad \log N}{k} \eta_d^k &\leq \left( \frac{c_e Ad \log N}{k} \right)^k \eta_d^k = \left( \eta_d \frac{c_e 12d \log N}{0.99(1 - \delta) \log N} \right)^k \\ &< \left( \eta_d \frac{70d}{0.99(1 - \delta)} \right)^k = \exp_2 \left( -\frac{dk}{0.98(1 - \delta)} \right) \leq N^{-1.01d}. \end{aligned}$$

Thus, again by the union bound, the probability that  $F_T$  has some clause that has more than  $k$  literals in  $\mathbf{S}$  is less than 0.9. Therefore, with some positive probability some  $\mathbf{S}$  (among randomly generated ones) satisfies the theorem.

Note that each assignment to variables in  $\mathbf{X} - \mathbf{S}$  yields a disjoint partial assignment  $\rho'$  of  $F_T$ . Thus, among them there should be some  $\rho'$  that has at least the sat. ratio of  $F_T$ , which is at least  $0.99 \cdot 2^{-2N^\delta}$ .  $\square$

We summarize our analysis and prove the theorem. For a given formula  $F$ , we define  $\rho_1$  and  $\rho_2$  as stated in Lemma 2 and Lemma 3 respectively. We use  $A = 12$  as mentioned above. Then we can guarantee that the resulting formula  $F' = F|\rho_2 \circ \rho_1$  has at least  $N' = 0.99\eta_d N/2$  variables, which are the variables in the set  $\mathbf{S}$  that  $\rho_2$  keeps unassigned among variables in  $F_T = F|\rho_1$ . Note also that  $F'$  consists of clauses of width  $\leq k = 0.99(1 - \delta) \log N$  and has sat. assignment ratio  $Q \geq 0.99 \cdot 2^{-2N^\delta}$ . Hence we apply Lemma 1 to this formula to show the existence of some partial assignment  $\rho_3$  (to  $F'$ ) of size at most

$$4c_e 2^k \log Q^{-1} \leq 4c_e N^{0.99(1-\delta)} \cdot (2N^\delta - \log 0.99) \leq 8.01c_e N^{1-0.01(1-\delta)},$$

which is smaller than  $N'/2$  for sufficiently large  $N$ . Thus, by defining  $\hat{\rho} = \rho_3 \circ \rho_2 \circ \rho_1$ , we have a satisfying partial assignment that keeps at least

$$\frac{N'}{2} = \frac{0.99 \cdot 0.99(1 - \delta)}{2 \cdot 70d} \exp_2 \left( -\frac{d}{0.98(1 - \delta)} \right) \cdot N \geq \frac{1 - \delta}{cd} \exp_2 \left( -\frac{(1 + o(1))d}{1 - \delta} \right) \cdot N$$

variables unassigned for some constant  $c > 0$ . This gives the desired upper bound to the size of our defined partial assignment  $\hat{\rho}$ .

### 4 A Lower Bound

We move on to the proof of Theorem 2. The idea is relatively easy. For any  $\delta$ ,  $0 < \delta < 1$ , and  $d \geq 1$ , consider  $\alpha$  satisfying (2) of Theorem 2. To be concrete, let us assume that  $\alpha = (d - 1.01)/(d - \delta)$ . Let  $\Pi$  be the set of partial assignments fixing  $\alpha N$  variables. Our goal is to show  $F$  that satisfies the conditions (i) and (ii) of the theorem and (iii) that is satisfied by no  $\rho \in \Pi$ .

We define  $F$  randomly as the conjunction of at most  $N^d$  random clauses chosen independently. Roughly speaking, each clause is a disjunction of approximately  $2s$  randomly chosen literals. The parameter  $s$  is chosen large enough to guarantee that each clause is satisfiable with a certain probability so that  $F$ 's sat. assignment ratio exceeds  $\exp_2(-N^\delta)$  with probability larger than some  $p$ . On the other hand, we keep  $s$  small enough so that each clause is satisfied with relatively small probability by fixing values of at most  $\alpha N$  variables, thereby ensuring that  $F|\rho = 1$  for some partial assignments  $\rho \in \Pi$  with probability  $\ll p$ . Then with the probabilistic argument, we can show the existence of our target Boolean formula  $F$ .

We start our detailed explanation with a precise way to generate a random clause. For some parameter  $s$  defined below, we consider the following way to

generate a random clause: For each  $i \in [N]$  independently, we select  $X_i$  as a literal of the clause with prob.  $s/N$ , select  $\bar{X}_i$  as a literal of the clause with prob.  $s/N$ , and discard  $X_i$  with prob.  $1 - 2s/N$ . The resulting clause is just the disjunction of the selected literals. In the following claim, we assume that  $C$  is a random clause obtained by this random clause generation. We fix  $s$  by  $s = (d - \delta) \ln N + 1$ . Then we have  $0 < s/N < 1$ ; hence, we can use  $s/N$  as a parameter for our random clause generation.

**Claim 1.** *For any assignment  $\mathbf{a} \in \{0, 1\}^N$  and any partial assignment  $\rho \in \Pi$ , we have  $\Pr_C[C(\mathbf{a}) = 0] \leq e^{-s}$ , and  $\Pr_C[C|\rho \neq 1] \geq 0.99e^{-s\alpha}$ .*

For generating a random formula  $F$  we iterate this random clause generation procedure independently for  $N^d$  times and define  $F$  as the conjunction of obtained clauses. In the following analysis, we use  $F$  as a random variable denoting a random formula generated in this way. We define  $p = \exp_2(-N^\delta)$ , and by the following two claims, we can show the probability that  $F$  satisfies the conditions of the theorem is at least, say,  $0.9p > 0$ , thereby proving the existence of the desired formula.

**Claim 2.**  $\Pr_F[\text{sat.ratio}(F) \geq p] \geq \exp_2(-N^\delta) (= p)$ .

**Claim 3.**  $\Pr_F[\exists \rho \in \Pi [F|\rho = 1]] \leq (3e^{-2})^N < 0.1p$  for sufficiently large  $N$ .

## 5 Algorithmic Version

In this section we explain the proof of Theorem 3. Due to the space limitation, we only give rough explanation of our proof.

The key tool is to use an algorithmic version of the Lovász Local Lemma, which has been improved greatly [2, 7, 8]. Our idea is simple. We show a sub-exponential-time deterministic algorithm that reduces our task to the CNF-SAT problem and use an algorithmic version of the Lovász Local Lemma. Here we use the version<sup>3</sup> reported in [2].

We specify our target problem and state the lemma in a slightly simpler way. Consider any sufficiently large  $N'$ , and let  $\mathcal{F}_{N'}$  denote the set of CNF formulas over  $N'$  Boolean variables with at most  $(N')^2$  clauses. The lemma gives an algorithm that finds a sat. assignment for any formula in  $\mathcal{F}_{N'}$  satisfying a certain condition. Let  $F'$  be any given formula in  $\mathcal{F}_{N'}$ . Consider a random assignment to its  $N'$  variables, and for each clause  $C$  of  $F'$ , let  $E_C$  denote an event that  $C$  becomes false by the assignment. Our goal (and the task of our algorithm) is to find an assignment avoiding  $E_C$  for all clauses  $C$  of  $F'$ , that is, to find a sat. assignment for this  $F' \in \mathcal{F}_{N'}$ . Let  $\Gamma(C)$  be the set of clauses

<sup>3</sup> In their paper, as a typical application of the lemma, an efficient deterministic algorithm is shown for  $k$ -CNF formulas with no variable appearing in many clauses. This may be used in our situation; but here we go back to the original lemma to confirm that our parameter choice works.

that shares some variable with  $C$ ; note that  $E_C$  is independent from  $E_{C'}$  for any  $C' \notin \Gamma(C)$ . In the lemma we consider some mapping  $x$  (for this  $F'$ ); by using this  $x$ , we also define  $x'(E_C)$  by

$$x'(E_C) = x(E_C) \prod_{C' \in \Gamma(C)} (1 - x(E_{C'})).$$

Now we state the following algorithmic version of the Lovász Local Lemma [2].

**Lemma 4.** *For any  $y > 0$ , there exists a deterministic algorithm that takes any  $F' \in \mathcal{F}_{N'}$  for any  $N' \geq 1$  as input, and runs in time  $O((N')^{c_{LLL}/y})$  yielding some sat. assignment of  $F'$  if we can define some mapping  $x$  for  $F'$  that satisfies*

$$\Pr[E_C] \leq x'(E_C)^{1+y}, \quad x(E_C) < 1/2, \quad \text{and} \quad x'(E_C) \geq (N')^{-1} \quad (4)$$

for all its clauses  $C$ , where  $c_{LLL} > 0$  is a constant independent from  $y$  and  $N'$ .

In the following, we show some algorithmic way to transform  $F$  to another formula  $F'$ ; we then apply this lemma to  $F'$  to obtain its sat. assignment, which can be used to define our desired partial assignment. Here, in order to explain requirements for  $F'$ , we consider some rough strategy for defining  $x$  for  $F'$  to satisfy the conditions of (4). For any clause  $C$  of  $F'$ , we have  $\Pr[E_C] = 2^{-|C|}$ . Thus, it is natural to define  $x(E_C) \approx 2^{-|C|}$ . Then we need to require that  $|C|$  is not so small, that is,  $C$  is not “very narrow” to satisfy the first and the third conditions of (4). We need, for example,  $|C| \geq \log N'$ . Also in order to avoid the situation where  $x'(E_C)$  gets too small compared with  $x(E_C)$ , we need to require that  $|\Gamma(C)|$  is not so large, that is,  $C$  is not so “popular.” When constructing  $F'$ , we need to consider these two requirements.

Now we explain the definition of our target sat. partial assignment  $\hat{\rho}$ . First let us fix input related parameters. Let  $F$  be any given CNF formula satisfying the condition of the theorem with parameters  $\delta$  and  $\varepsilon$ , and let  $\alpha$  be the constant defined by (3). Let  $\gamma = 1 - (\delta + \varepsilon)$ , which is positive, though potentially small constant<sup>4</sup>. Fix  $F$ ,  $\delta$ ,  $\varepsilon$ ,  $\gamma$ , and  $\alpha$  from now on. We define  $\hat{\rho}$  in three stages. In the first stage, a partial assignment  $\rho_1$  is defined in a way similar to the first stage in the proof of Theorem 1. In the second stage, we convert  $F$  to  $F'$  by removing some number of variables randomly from  $F|\rho_1$  so that it is still satisfiable and we can use the algorithm of the above lemma to find one of its sat. assignments. Then in the third stage, we use the above algorithm to compute a sat. assignment of  $F'$ . Note that this *complete* assignment to  $F'$  can be regarded as a sat. partial assignment  $\rho_2$  of  $F|\rho_1$  that leaves all (and only) removed variables unassigned; we define our final assignment  $\hat{\rho}$  by  $\hat{\rho} = \rho_2 \circ \rho_1$ . The partial assignment  $\rho_1$  is defined to satisfy the following two requirements: (a')  $F|\rho_1$  has no “narrow”

---

<sup>4</sup> For simplicity, we assume that  $\gamma < 0.5$ . The case where  $\gamma \geq 0.5$  can be analyzed similarly with different setting for our technical parameters  $b$  and  $B$ .

clause, and (b)  $F|\rho_1$  has no “popular” literal. Then from (a’) we can show that (a)  $F'$  has no “very narrow” clause with high probability after removing some number of variables. By using (a) together with (b), we can satisfy the conditions of (4). (Note that  $F'$  clearly satisfies (b) if  $F|\rho_1$  does.) More specifically with parameters  $\ell$  and  $L$  defined below, we say that a clause is *narrow* if its width is less than  $\ell$ , and a literal is *popular* (in a currently considered CNF formula) if it appears in more than  $L$  clauses.

$$\ell = b(1 - \delta) \log N, \quad \text{and} \quad L = N^{(1-b\gamma)(1-\delta)}.$$

Here  $b$  is some constant  $b < 1$ ; for example, we can show that the whole proof goes through by choosing  $b$  by  $1 - 0.4\gamma$ .

We can show that the assignment  $\hat{\rho} = \rho_2 \circ \rho_1$  defined above leaves  $\Omega(N)$  variables unassigned as desired. First, we show that  $\rho_1$  leaves some  $(1 - o(1))N$  variables unassigned. Like the previous  $\rho_1$ , our  $\rho_1$  is defined by using a sequence  $\sigma_1, \sigma_2, \dots$  of very short partial assignments defined step by step. Here we need to eliminate narrow clauses and popular literals. To eliminate each narrow clause, we fix the values of all literals in the clause. We show that the sat. ratio increases a good amount by using an appropriate assignment to those literals. Hence, the number of applying very short partial assignments of this type is limited (because otherwise, the sat. ratio exceeds 1). On the other hand, we eliminate popular literals (here w.r.t. all clauses in the current formula) in the same way as before, and by the same reasoning, we can bound the number of applying this step. Altogether we can show that there exists some  $\rho_1$  that satisfies both (a’) and (b) by fixing at most  $O(N^\beta)$  variables, where  $\beta < 1$  is the constant specified in the theorem. Then we show that with high probability one can remove  $\Omega(N)$  variables from  $F|\rho_1$  while keeping both (a) and (b) so that we can apply the above lemma to find an assignment satisfying  $F'$ .

We can give a deterministic algorithm implementing these three stages. First for finding the best  $\rho_1$ , the algorithm tries all possible candidate partial assignments. We can show that this brute force search is conducted in  $\tilde{O}(2^{N^\beta})$  steps. Next, the random removable of variables of the second stage can be derandomized in polynomial-time by the standard method of conditional probabilities. Thus,  $F'$  is obtained deterministically in  $\tilde{O}(2^{N^\beta})$ -time. Then the above lemma guarantees that one of its sat. assignment is computed in  $O(N^{O(1/y)})$ -time. Since the last time bound is subsumed by  $\tilde{O}(2^{N^\beta})$ , we can conclude that  $\hat{\rho} = \rho_2 \circ \rho_1$  is deterministically computable in  $\tilde{O}(2^{N^\beta})$ -time. This is the outline of our deterministic algorithm for computing  $\hat{\rho}$ .

**Acknowledgments.** This work was started from the discussion at the workshop on Computational Complexity at the Banff International Research Station for Mathematical Innovation and Discovery (BIRS), 2013. The first author is supported in part by an NSF postdoctoral research fellowship. The second author is supported in part by the ELC project (MEXT KAKENHI Grant No. 24106008).

## References

1. Ajtai, M.:  $\Sigma_1^1$ -formula on finite structures. *Ann. Pure. Appl. Logic* **24**, 1–48 (1983)
2. Chandrasekaran, K., Goyal, N., Haeupler, B.: Deterministic algorithms for the Lovász Local Lemma. In: *Proc. of the 21st Annual ACM-SIAM Sympos. on Discrete Algorithms, SODA 2010*, pp. 992–1004. SIAM (2010)
3. De, A., Etesami, O., Trevisan, L., Tulsiani, M.: Improved pseudorandom generators for depth 2 circuits. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) *APPROX and RANDOM 2010*. LNCS, vol. 6302, pp. 504–517. Springer, Heidelberg (2010)
4. Håstad, J.: Computational limitations of small depth circuits, Ph.D. thesis, Massachusetts Institute of Technology (1987)
5. Hirsch, E.: A fast deterministic algorithm for formulas that have many satisfying assignments. *Logic Journal of the IGPL* **6**(1), 59–71 (1998)
6. Kane, D., Watanabe, O.: A short implicant of CNFs with relatively many satisfying assignments, *ECCC TR13-176* (December 2013) (revised April 2014)
7. Moser, R.: A constructive proof of the Lovász local lemma. In: *Proc. of the 41st Annual ACM Sympos. on Theory of Computing, STOC 2009*, pp. 343–350 (2009)
8. Moser, R., Tardos G.: A constructive proof of the general Lovász local lemma. *J. ACM* **57**(2) (2010)

# On the Computational Complexity of Vertex Integrity and Component Order Connectivity

Pål Grønås Drange<sup>(✉)</sup>, Markus Sortland Dregi, and Pim van 't Hof

Department of Informatics, University of Bergen, Bergen, Norway  
{pal.drange,markus.dregi,pim.vanthof}@ii.uib.no

**Abstract.** The WEIGHTED VERTEX INTEGRITY (wVI) problem takes as input an  $n$ -vertex graph  $G$ , a weight function  $w : V(G) \rightarrow \mathbb{N}$ , and an integer  $p$ . The task is to decide if there exists a set  $X \subseteq V(G)$  such that the weight of  $X$  plus the weight of a heaviest component of  $G - X$  is at most  $p$ . Among other results, we prove that:

- (1) wVI is NP-complete on co-comparability graphs, even if each vertex has weight 1;
- (2) wVI can be solved in  $O(p^{p+1}n)$  time;
- (3) wVI admits a kernel with at most  $p^3$  vertices.

Result (1) refutes a conjecture by Ray and Deogun (J. Combin. Math. Combin. Comput. 16: 65–73, 1994) and answers an open question by Ray et al. (Ars Comb. 79: 77–95, 2006). It also complements a result by Kratsch et al. (Discr. Appl. Math. 77: 259–270, 1997), stating that the unweighted version of the problem can be solved in polynomial time on co-comparability graphs of bounded dimension, provided that an intersection model of the input graph is given as part of the input.

An instance of the WEIGHTED COMPONENT ORDER CONNECTIVITY (wCOC) problem consists of an  $n$ -vertex graph  $G$ , a weight function  $w : V(G) \rightarrow \mathbb{N}$ , and two integers  $k$  and  $\ell$ , and the task is to decide if there exists a set  $X \subseteq V(G)$  such that the weight of  $X$  is at most  $k$  and the weight of a heaviest component of  $G - X$  is at most  $\ell$ . In some sense, the wCOC problem can be seen as a refined version of the wVI problem. We obtain several classical and parameterized complexity results on the wCOC problem, uncovering interesting similarities and differences between wCOC and wVI. We prove, among other results, that:

- (4) wCOC can be solved in  $O(\min\{k, \ell\} \cdot n^3)$  time on interval graphs, while the unweighted version can be solved in  $O(n^2)$  time on this graph class;
- (5) wCOC is W[1]-hard on split graphs when parameterized by  $k$  or by  $\ell$ ;
- (6) wCOC can be solved in  $2^{O(k \log \ell)} n$  time;
- (7) wCOC admits a kernel with at most  $k\ell(k + \ell) + k$  vertices.

We also show that result (6) is essentially tight by proving that wCOC cannot be solved in  $2^{o(k \log \ell)} n^{O(1)}$  time, even when restricted to split graphs, unless the Exponential Time Hypothesis fails.

---

The research leading to these results has received funding from the Research Council of Norway, Bergen Research Foundation under the project Beating Hardness by Pre-processing and the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 267959.

## 1 Introduction

Motivated by a multitude of practical applications, many different vulnerability measures of graphs have been introduced in the literature over the past few decades. The vertex and edge connectivity of a graph, although undoubtedly being the most well-studied of these measures, often fail to capture the more subtle vulnerability properties of networks that one might wish to consider, such as the number of resulting components, the size of the largest or smallest component that remains, and the largest difference in size between any two remaining components. The two vulnerability measures we study in this paper, *vertex integrity* and *component order connectivity*, take into account not only the number of vertices that need to be deleted in order to break a graph into pieces, but also the number of vertices in the largest component that remains.

The *vertex integrity* of an unweighted graph  $G$  is defined as  $\iota(G) = \min\{|X| + n(G - X) \mid X \subseteq V(G)\}$ , where  $n(G - X)$  is the number of vertices in the largest connected component of  $G - X$ . This vulnerability measure was introduced by Barefoot, Entringer and Swart [2] in 1987. For an overview of structural results on vertex integrity, including combinatorial bounds and relationships between vertex integrity and other vulnerability measures, we refer the reader to a survey on the subject by Bagga et al. [1]. We mention here only known results on the computational complexity of determining the vertex integrity of a graph.

The VERTEX INTEGRITY (VI) problem takes as input an  $n$ -vertex graph  $G$  and an integer  $p$ , and asks whether  $\iota(G) \leq p$ . This problem was shown to be NP-complete, even when restricted to planar graphs, by Clark, Entringer, and Fellows [6]. On the positive side, Fellows and Stueckle [8] showed that the problem can be solved in  $O(p^3pn)$  time, and is thus fixed-parameter tractable when parameterized by  $p$ . In the aforementioned survey, Bagga et al. [1] mention that VERTEX INTEGRITY can be solved in  $O(n^3)$  time when the input graph is a tree or a cactus graph. Kratsch, Kloks, and Müller [12] studied the computational complexity of determining the value of several vulnerability measures in classes of intersection graphs. Their results imply that VERTEX INTEGRITY can be solved in  $O(n^3)$  time on interval graphs, in  $O(n^4)$  time on circular-arc graphs, and in  $O(n^5)$  time on permutation graphs and trapezoid graphs. Kratsch et al. [12] also mention that the problem can be solved in  $O(n^{2d+1})$  time on co-comparability graphs of dimension at most  $d$ , provided that an intersection model of the input graph is given as part of the input.

Ray and Deogun [14] were the first to study the more general WEIGHTED VERTEX INTEGRITY (wVI) problem. This problem takes as input an  $n$ -vertex graph  $G$ , a weight function  $w : V(G) \rightarrow \mathbb{N}$ , and an integer  $p$ . The task is to decide if there exists a set  $X \subseteq V(G)$  such that the weight of  $X$  plus the weight of a heaviest component of  $G - X$  is at most  $p$ . Using a reduction from 0-1 KNAPSACK, Ray and Deogun [14] identified several graph classes on which the WEIGHTED VERTEX INTEGRITY problem is weakly NP-complete. In particular, their result implies that the problem is weakly NP-complete on trees, bipartite graphs, series-parallel graphs, and regular graphs, and therefore also on super-classes such as chordal graphs and comparability graphs. A common property of



these classes is that they contain graphs with arbitrarily many asteroidal triples and induced paths on five vertices; any graph class that does not have this property is not covered by the result Ray and Deogun. They conjectured that the WEIGHTED VERTEX INTEGRITY problem can be solved in polynomial time on co-comparability graphs, a well-known example of a class of graphs that do not contain asteroidal triples at all. More than a decade later, Ray et al. [15] presented a polynomial-time algorithm for WEIGHTED VERTEX INTEGRITY on interval graphs, a subclass of co-comparability graphs. In the same paper, they pointed out that the complexity of the problem on co-comparability graphs remained unknown.

We now turn our attention to the second vulnerability measure studied in this paper. For any positive integer  $\ell$ , the  $\ell$ -component order connectivity of a graph  $G$  is defined to be the cardinality of a smallest set  $X \subseteq V(G)$  such that  $n(G - X) < \ell$ . We refer to the survey by Gross et al. [10] for more background on this graph parameter. Motivated by the definitions of  $\ell$ -component order connectivity and the WEIGHTED VERTEX INTEGRITY problem, we introduce the WEIGHTED COMPONENT ORDER CONNECTIVITY (wCOC) problem. This problem takes as input a graph  $G$ , a weight function  $w : V(G) \rightarrow \mathbb{N}$ , and two integers  $k$  and  $\ell$ . The task is to decide if there exists a set  $X \subseteq V(G)$  such that the weight of  $X$  is at most  $k$  and the weight of a heaviest component of  $G - X$  is at most  $\ell$ . Observe that the WEIGHTED COMPONENT ORDER CONNECTIVITY problem can be interpreted as a more refined version of WEIGHTED VERTEX INTEGRITY. We therefore find it surprising that, to the best of our knowledge, the WEIGHTED COMPONENT ORDER CONNECTIVITY problem has not yet been studied in the literature. We do however point out that the techniques described by Kratsch et al. [12] yield polynomial-time algorithms for the unweighted version of the problem on interval graphs, circular-arc graphs, permutation graphs, and trapezoid graphs, and that very similar problems have received some attention recently [3, 10].

**Our Contribution.** In Section 2, we present our results on VERTEX INTEGRITY and WEIGHTED VERTEX INTEGRITY. We show that VI is NP-complete on co-bipartite graphs, and hence on co-comparability graphs. This refutes the aforementioned conjecture by Ray and Deogun [14] and answers an open question by Ray et al. [15]. It also forms an interesting contrast with the result by Kratsch et al. [12] stating that VI can be solved in  $O(n^{2d+1})$  time on co-comparability graphs of dimension at most  $d$  if an intersection model is given as part of the input. We also show that even though VI can be solved in linear time on split graphs, the problem remains NP-complete on chordal graphs. Interestingly, we prove that unlike the unweighted variant of the problem, the wVI problem is NP-complete when restricted to split graphs; observe that this does not follow from the aforementioned hardness result by Ray and Deogun [14], as split graphs do not contain induced paths on five vertices.

Recall that Fellows and Stueckle [8] showed that VI can be solved in  $O(p^{3p}n)$  time on general graphs. We strengthen this result by showing that even the wVI

problem can be solved in  $O(p^{p+1}n)$  time. We also show that wVI admits a kernel with at most  $p^3$  vertices, each having weight at most  $p$ .

**Table 1.** An overview of the classical complexity results proved in this paper. Previously known results are given with a reference.

	VI	wVI	COC	wCOC
general	NPc [6]	NPc [6]	NPc [6]	NPc [6]
co-bipartite	NPc	NPc	NPc	NPc
chordal	NPc	NPc	NPc	NPc
split	$O(n + m)$ [13]	NPc	NPc	NPc
interval	$O(n^3)$ [12]	$O(n^6 \log n)$ [15]	$O(n^2)$	$O(\min\{k, \ell\} \cdot n^3)$
complete	$O(n)$	$O(n)$	$O(n)$	weakly NPc

Section 3 contains our results on COMPONENT ORDER CONNECTIVITY and WEIGHTED COMPONENT ORDER CONNECTIVITY. The observation that there is a polynomial-time Turing reduction from VI to COC implies that the latter problem cannot be solved in polynomial time on any graph class for which VI is NP-complete, unless  $P=NP$ . We prove that wCOC is weakly NP-complete already on complete graphs, while the unweighted variant of the problem, which is trivial on complete graphs, remains NP-complete when restricted to split graphs. We find the latter result particularly interesting in light of existing polynomial-time algorithms for computing similar (unweighted) vulnerability measures of split graphs, such as toughness [16], vertex integrity, scattering number, tenacity, and rupture degree [13]. To complement our hardness results, we present a pseudo-polynomial-time algorithm that solves the wCOC problem in  $O(\min\{k, \ell\} \cdot n^3)$  time on interval graphs. We then modify this algorithm to solve the unweighted version of the problem in  $O(n^2)$  time on interval graphs, thereby improving the  $O(n^3)$ -time algorithm that follows from the results by Kratsch et al. [12]. Observe that the aforementioned hardness results rule out the possibility of solving wCOC in polynomial time on interval graphs or in pseudo-polynomial time on split graphs.

In Section 3, we also completely classify the parameterized and kernelization complexity of COC and wCOC on general graphs with respect to the parameters  $k$ ,  $\ell$ , and  $k + \ell$ . We first observe that both problems are para-NP-hard when parameterized by  $\ell$  due to the fact that COC is equivalent to VERTEX COVER when  $\ell = 1$ . We then prove that if we take either  $k$  or  $\ell$  to be the parameter, then COC is W[1]-hard even on split graphs. On the positive side, we show that wCOC becomes fixed-parameter tractable when parameterized by  $k + \ell$ . We present an algorithm for solving the problem in time  $2^{O(k \log \ell)}n$  time, before proving that the problem cannot be solved in time  $2^{o(k \log \ell)}n^{O(1)}$  unless the Exponential Time Hypothesis fails. Finally, we show that wCOC admits a polynomial kernel with at most  $k\ell(k + \ell) + k$  vertices, where each vertex has weight at most  $k + \ell$ .

**Notation and Terminology.** For a reader not familiar with graph classes and parameterized complexity, we refer to the full version of the paper [7] for additional terminology.

Let  $G$  be a graph and  $w : V(G) \rightarrow \mathbb{N} = \{0, 1, \dots\}$  a weight function on the vertices of  $G$ . The weight of a subset  $X \subseteq V(G)$  is defined as  $w(X) = \sum_{v \in X} w(v)$ . We define  $w_{cc}(G)$  to be the weight of a heaviest component of  $G$ , i.e.,  $w_{cc}(G) = \max\{w(V(G_i)) \mid 1 \leq i \leq r\}$ , where  $G_1, \dots, G_r$  are the components of  $G$ . The *weighted vertex integrity* of  $G$  is defined as

$$\iota(G) = \min\{w(X) + w_{cc}(G - X) \mid X \subseteq V(G)\},$$

where  $G - X$  denotes the graph obtained from  $G$  by deleting all the vertices in  $X$ . Any set  $X \subseteq V(G)$  for which  $w(X) + w_{cc}(G - X) = \iota(G)$  is called an  $\iota$ -set of  $G$ . We consider the following two decision problems:

**WEIGHTED VERTEX INTEGRITY (wVI)**

*Instance:* A graph  $G$ , a weight function  $w : V(G) \rightarrow \mathbb{N}$ , and an integer  $p$ .

*Question:* Is  $\iota(G) \leq p$ ?

**WEIGHTED COMPONENT ORDER CONNECTIVITY (wCOC)**

*Instance:* A graph  $G$ , a weight function  $w : V(G) \rightarrow \mathbb{N}$ , and two integers  $k$  and  $\ell$ .

*Question:* Is there a set  $X \subseteq V(G)$  with  $w(X) \leq k$  such that  $w_{cc}(G - X) \leq \ell$ ?

The unweighted versions of these two problems, where  $w(v) = 1$  for every vertex  $v \in V(G)$ , are called VERTEX INTEGRITY (VI) and COMPONENT ORDER CONNECTIVITY (COC), respectively. Recall that the *vertex integrity* of an unweighted graph  $G$  is defined as  $\iota(G) = \min\{|X| + n(G - X) \mid X \subseteq V(G)\}$ , where  $n(G - X)$  is the number of vertices in the largest connected component of  $G - X$ .

## 2 Vertex Integrity

As mentioned in the introduction, Ray et al. [15] asked whether WEIGHTED VERTEX INTEGRITY can be solved in polynomial time on co-comparability graphs. We show that this is not the case, unless  $P = NP$ . In fact, we prove a much stronger result in Theorem 1 below by showing NP-completeness of an easier problem (VERTEX INTEGRITY) on a smaller graph class (co-bipartite graphs).

**Theorem 1.** VERTEX INTEGRITY is NP-complete on co-bipartite graphs.

*Proof.* The problem is clearly in NP. To show that it is NP-hard, we give a polynomial-time reduction from the BALANCED COMPLETE BIPARTITE SUBGRAPH problem. This problem, which is known to be NP-complete [9], takes as input a bipartite graph  $G = (A, B, E)$  and an integer  $k \geq 1$ , and asks whether there exist subsets  $A' \subseteq A$  and  $B' \subseteq B$  such that  $|A'| = |B'| = k$  and  $G[A' \cup B']$  is a complete bipartite graph. Let  $(G, k)$  be an instance of BALANCED COMPLETE BIPARTITE SUBGRAPH, where  $G = (A, B, E)$  is a bipartite graph on  $n$  vertices.

We claim that  $(G, k)$  is a yes-instance of BALANCED COMPLETE BIPARTITE SUBGRAPH if and only if  $(\overline{G}, n - k)$  is a yes-instance of VERTEX INTEGRITY.

Suppose there exist subsets  $A' \subseteq A$  and  $B' \subseteq B$  such that  $|A'| = |B'| = k$  and  $A' \cup B'$  induces a complete bipartite subgraph in  $G$ . Observe that in  $\overline{G}$ , both  $A'$  and  $B'$  are cliques, and there is no edge between  $A'$  and  $B'$ . Hence, if we delete all the vertices in  $V(\overline{G}) \setminus (A' \cup B')$  from  $\overline{G}$ , the resulting graph has exactly two components containing exactly  $k$  vertices each. Since  $|V(\overline{G}) \setminus (A' \cup B')| = n - 2k$ , it holds that  $\iota(\overline{G}) \leq n - 2k + k = n - k$ , and hence  $(\overline{G}, n - k)$  is a yes-instance of VERTEX INTEGRITY.

For the reverse direction, suppose  $(\overline{G}, n - k)$  is a yes-instance of VERTEX INTEGRITY. Then there exists a subset  $X \subseteq V(\overline{G})$  such that  $|X| + n(\overline{G} - X) \leq n - k$ . The assumption that  $k \geq 1$  implies that  $\overline{G} - X$  is disconnected, as otherwise  $|X| + n(\overline{G} - X) = V(\overline{G}) = n$ . Let  $A' = A \setminus X$  and  $B' = B \setminus X$ . Since  $\overline{G}$  is co-bipartite, both  $A'$  and  $B'$  are cliques. Moreover, since  $\overline{G} - X$  is disconnected, there is no edge between  $A'$  and  $B'$ . Hence,  $\overline{G}[A']$  and  $\overline{G}[B']$  are the two components of  $\overline{G} - X$ . Without loss of generality, suppose that  $|A'| \geq |B'|$ . Then  $|B'| = n - (|X| + |A'|) = n - (|X| + n(\overline{G} - X)) \geq n - (n - k) = k$  and hence  $|A'| \geq |B'| \geq k$ . This, together with the observation that  $A' \cup B'$  induces a complete bipartite subgraph in  $G$ , implies that  $(G, k)$  is a yes-instance of BALANCED COMPLETE BIPARTITE SUBGRAPH.  $\square$

Ray and Deogun [14] proved that WEIGHTED VERTEX INTEGRITY is NP-complete on any graph class that satisfies certain conditions. Without explicitly stating these (rather technical) conditions here, let us point out that any graph class satisfying these conditions must contain graphs with arbitrarily many asteroidal triples and induced paths on five vertices. Theorem 1 shows that neither of these two properties is necessary to ensure NP-completeness of WEIGHTED VERTEX INTEGRITY, since co-bipartite graphs contain neither asteroidal triples nor induced paths on five vertices.

In Theorem 2 below, we show that WEIGHTED VERTEX INTEGRITY is NP-complete on split graphs. Since split graphs do not contain induced paths on five vertices, this graph class is not covered by the aforementioned hardness result of Ray and Deogun [14].

**Lemma 1.** (★)<sup>1</sup> *For every graph  $G$  and weight function  $w : V(G) \rightarrow \mathbb{N}$ , there exists an  $\iota$ -set  $X$  that contains no simplicial vertices of  $G$ .*

Given a graph  $G$ , the *incidence split graph* of  $G$  is the split graph  $G^* = (C^*, I^*, E^*)$  whose vertex set consists of a clique  $C^* = \{v_x \mid x \in V(G)\}$  and an independent set  $I^* = \{v_e \mid e \in E(G)\}$ , and where two vertices  $v_x \in C^*$  and  $v_e \in I^*$  are adjacent if and only if vertex  $x$  is incident with edge  $e$  in  $G$ . The following lemma will be used in the proofs of hardness results not only in this section, but also in Section 3.

<sup>1</sup> Due to page restrictions, proofs of results marked with a star have been deferred to the full version [7].

**Lemma 2.** (★) *Let  $G = (V, E)$  be a graph,  $G^* = (C^*, I^*, E^*)$  its incidence split graph, and  $k < |V|$  a non-negative integer. Then the following statements are equivalent:*

- (i)  $G$  has a clique of size  $k$ ;
- (ii) there exists a set  $X \subseteq C^*$  such that  $|X| \leq k$  and  $|X| + n(G^* - X) \leq |V| + |E| - \binom{k}{2}$ ;
- (iii) there exists a set  $X \subseteq C^*$  such that  $|X| \leq k$  and  $n(G^* - X) \leq |V| + |E| - \binom{k}{2} - k$ .

**Theorem 2.** WEIGHTED VERTEX INTEGRITY is NP-complete on split graphs.

*Proof.* We give a reduction from the NP-hard problem CLIQUE. Given an instance  $(G, k)$  of CLIQUE with  $n = |V(G)|$  and  $m = |E(G)|$ , we create an instance  $(G', w, p)$  of WEIGHTED VERTEX INTEGRITY as follows. To construct  $G'$ , we start with the incidence split graph  $G^* = (C^*, I^*, E^*)$  of  $G$ , and we add a single isolated vertex  $z$ . We define the weight function  $w$  by setting  $w(z) = n + m - \binom{k}{2} - k$  and  $w(v) = 1$  for every  $v \in V(G') \setminus \{z\}$ . Finally, we set  $p = n + m - \binom{k}{2}$ . For convenience, we assume that  $k < n$ .

We claim that  $G$  has a clique of size  $k$  if and only if  $\iota(G') \leq p$ . Since  $G'$  is a split graph and all the vertex weights are polynomial in  $n$ , this suffices to prove the theorem.

First suppose  $G$  has a clique  $S$  of size  $k$ . By Lemma 2, there exists a set  $X \subseteq C$  such that  $|X| \leq k$  and  $n(G^* - X) \leq n + m - \binom{k}{2} - k$ . Since  $w(z) = n + m - \binom{k}{2} - k$  and every other vertex in  $G'$  has weight 1, it follows that  $w_{cc}(G' - X) = n + m - \binom{k}{2} - k$ . Consequently,  $w(X) + w_{cc}(G' - X) \leq n + m - \binom{k}{2} = p$ , so we conclude that  $\iota(G') \leq p$ .

For the reverse direction, suppose  $\iota(G') \leq p$ , and let  $X \subseteq V(G')$  be an  $\iota$ -set of  $G'$ . Due to Lemma 1, we may assume that  $X \subseteq C$ . We claim that  $|X| \leq k$ . For contradiction, suppose  $|X| \geq k + 1$ . Then  $w(X) = |X| \geq k + 1$  and  $w_{cc}(G' - X) \geq w(z) = p - k$ . This implies that  $\iota(G') = w(X) + w_{cc}(G' - X) \geq p + 1$ , yielding the desired contradiction. Now let  $H$  be the component of  $G' - X$  containing the clique  $C \setminus X$ . The fact that every vertex in  $V(G') \setminus \{z\}$  has weight 1 and the assumption that  $k < n$  imply that  $|V(H)| = n(G^* - X) = w_{cc}(G^* - X)$ . Now observe that  $|X| + n(G^* - X) \leq |X| + \max\{w(z), w_{cc}(G^* - X)\} = \iota(G') \leq p = n + m - \binom{k}{2}$ . We can therefore invoke Lemma 2 to conclude that  $G$  has a clique of size  $k$ . □

The following result, previously obtained by Li et al. [13], is an easy consequence of Lemma 1. Theorem 4 below shows that this result is in some sense best possible.

**Theorem 3** ([13]). VERTEX INTEGRITY can be solved in linear time on split graphs.

**Theorem 4.** (★) VERTEX INTEGRITY is NP-complete on chordal graphs.

Recall that Fellows and Stueckle [8] proved that VERTEX INTEGRITY can be solved in time  $O(p^{3p}n)$ . Their arguments can be slightly strengthened to yield the following result.

**Theorem 5.** (★) WEIGHTED VERTEX INTEGRITY can be solved in  $O(p^{p+1}n)$  time.

We prove that the problem admits a polynomial kernel with respect to parameter  $p$ .

**Theorem 6.** (★) WEIGHTED VERTEX INTEGRITY admits a kernel with at most  $p^3$  vertices, where each vertex has weight at most  $p$ .

### 3 Component Order Connectivity

It is easy to see that  $(G, p)$  is a yes-instance of VERTEX INTEGRITY if and only if there exist non-negative integers  $k$  and  $\ell$  with  $k + \ell = p$  such that  $(G, k, \ell)$  is a yes-instance of COMPONENT ORDER CONNECTIVITY. Hence, any instance  $(G, p)$  of VERTEX INTEGRITY can be solved by making at most  $p$  calls to an algorithm solving COMPONENT ORDER CONNECTIVITY, implying that COMPONENT ORDER CONNECTIVITY cannot be solved in polynomial time on any graph class for which VERTEX INTEGRITY is NP-complete, unless  $P=NP$ .

Our next two results identify graph classes for which wCOC and COC are strictly harder than wVI and VI, respectively.

**Theorem 7.** (★) WEIGHTED COMPONENT ORDER CONNECTIVITY is weakly NP-complete on complete graphs.

**Theorem 8.** (★) COMPONENT ORDER CONNECTIVITY is NP-complete on split graphs.

We now present a pseudo-polynomial algorithm, called wCOC, that solves WEIGHTED COMPONENT ORDER CONNECTIVITY in  $O(kn^3)$  time on interval graphs. We refer to Figure 1 for pseudocode of the algorithm.

Given an instance  $(G, w, k, \ell)$ , where  $G$  is an interval graph, the algorithm first removes every vertex of weight 0. It then computes a *clique path* of  $G$ , i.e., an ordering  $K_1, \dots, K_t$  of the maximal cliques of  $G$  such that for every vertex  $v \in V(G)$ , the maximal cliques containing  $v$  appear consecutively in this ordering. Since  $G$  is an interval graph, such an ordering exists and can be obtained in  $O(n^2)$  time [5]. For convenience, we define two empty sets  $K_0$  and  $K_{t+1}$ . The algorithm now computes the set  $S_i = K_i \cap K_{i+1}$  for every  $i \in \{0, \dots, t\}$ . Observe that  $S_0$  and  $S_t$  are both empty by construction, and that the non-empty sets among  $S_1, \dots, S_{t-1}$  are exactly the minimal separators of  $G$  (see, e.g., [11]). For every  $q \in \{0, \dots, t+1\}$ , we define  $G_q = G[\bigcup_{i=0}^q K_i]$ . Also, for any two integers  $i, j$  with  $0 \leq i < j \leq t$ , the algorithm computes the set  $V_{i,j} = V(G[\bigcup_{p=i+1}^j K_p \setminus (S_i \cup S_j)])$ . Informally speaking, the set  $V_{i,j}$  consists of the vertices of  $G$  that lie “in between” separators  $S_i$  and  $S_j$ .

```

Algorithm wCOC
Input: An instance  $(G, w, k, \ell)$  of WEIGHTED COMPONENT ORDER
CONNECTIVITY, where  $G$  is an interval graph
Output: “yes” if  $(G, w, k, \ell)$  is a yes-instance, and “no” otherwise

Remove every vertex of weight 0 from  $G$ 
Construct  $K_0, \dots, K_{t+1}$ 
Construct  $S_0, \dots, S_t$ 
Construct  $V_{i,j}$  for every  $0 \leq i < j \leq t$ 

Set all elements of  $dp$  to  $k + 1$ 
Set  $dp[0] = 0$ 
for  $j$  from 1 to  $t$  do
    for  $i$  from  $j - 1$  to 0 do
        Let  $v_1, \dots, v_{|V_{i,j}|}$  be the vertices of  $V_{i,j}$ 
        Let  $w_p = w(v_p)$  for every  $p \in \{1, \dots, |V_{i,j}|\}$ 
        if  $w(V_{i,j}) \leq k + \ell$  then
            Let  $I = \text{MinSup}((w_1, \dots, w_{|V_{i,j}|}), w(V_{i,j}) - \ell)$ 
            Let  $Y_{i,j} = \{v_p \in V_{i,j} \mid p \in I\}$ 
             $dp[j] = \min \begin{cases} dp[j] \\ dp[i] + w(Y_{i,j}) + w(S_j \setminus S_i) \end{cases}$ 
        end
    end
end
return “yes” if  $dp[t] \leq k$ , and “no” otherwise
    
```

**Fig. 1.** Pseudocode of the algorithm wCOC that solves the WEIGHTED COMPONENT ORDER CONNECTIVITY problem on interval graphs in  $O(kn^3)$  time

Let us give some intuition behind the next phase of the algorithm. Suppose  $(G, w, k, \ell)$  is a yes-instance of WEIGHTED COMPONENT ORDER CONNECTIVITY, and let  $X$  be a solution for this instance. Generally speaking,  $X$  fully contains some minimal separators of  $G$  whose removal is necessary to break the graph into pieces, as well as additional vertices that are deleted from these pieces with the sole purpose of decreasing the weight of each piece to at most  $\ell$ . The constructed clique path  $K_1, \dots, K_t$  corresponds to a linear order of the minimal separators  $S_1, \dots, S_{t-1}$  of  $G$ . We will use this linear structure to find a minimum solution by doing dynamic programming over the minimal separators of  $G$ .

For every  $q \in \{0, \dots, t\}$ , let  $k_q$  denote the smallest integer such that there exists a set  $X \subseteq V(G)$  satisfying the following three properties:

- $w(X) = k_q$ ;
- $S_q$  is a subset of  $X$ ;
- $X$  is a solution for the instance  $(G_q, w, k_q, \ell)$ .

In other words,  $X$  is a “cheapest” solution for  $(G_q, w, k_q, \ell)$  that fully contains the minimal separator  $S_q$ . The algorithm now constructs an array  $\text{dp}$  with  $t + 1$  entries, each of which is an integer from  $\{0, \dots, k + 1\}$ . Initially, all the elements of the array are set to  $k + 1$ . For any  $q \in \{0, \dots, t\}$ , we say that the entry  $\text{dp}[q]$  has *reached optimality* if

$$\text{dp}[q] = \begin{cases} k_q & \text{if } k_q \leq k \\ k + 1 & \text{otherwise.} \end{cases}$$

Recall that  $S_t = \emptyset$  and that  $G_t = G$ . Hence, if  $\text{dp}[t]$  has reached optimality, then the input instance  $(G, w, k, \ell)$  is a yes-instance if and only if  $\text{dp}[t] \leq k$ .

The algorithm uses a subroutine **MinSup** that, given a multiset of  $r$  weights  $(w_1, \dots, w_r)$  and a target  $W$  such that  $\sum_{i=1}^r w_i \geq W$ , finds a set  $I \subseteq \{1, \dots, r\}$  such that  $\sum_{i \in I} w_i$  is minimized with respect to the constraint  $\sum_{i \in I} w_i \geq W$ . Note that this subroutine **MinSup** can be implemented to run in time  $O(Wr)$  using the classical dynamic programming algorithm for **SUBSET SUM**.

**Theorem 9.** (★) **WEIGHTED COMPONENT ORDER CONNECTIVITY** can be solved in  $O(\min\{k, \ell\} \cdot n^3)$  time on interval graphs.

**Theorem 10.** **COMPONENT ORDER CONNECTIVITY** can be solved in  $O(n^2)$  time on interval graphs.

*Proof.* We describe a modification of the algorithm **wCOC**, called **uCOC**, that solves the unweighted **COMPONENT ORDER CONNECTIVITY** problem in  $O(n^2)$  time on interval graphs. There are two reasons why the algorithm **wCOC** does not run in  $O(n^2)$  time: constructing all the sets  $V_{i,j}$  takes  $O(n^3)$  time in total, and each of the  $O(n^2)$  executions of the inner loop takes  $O(kn)$  time, which is the time taken by the subroutine **MinSup** to compute the set  $Y_{i,j}$  of vertices that are to be deleted.

Recall that for every  $j \in \{1, \dots, t\}$  and every  $i \in \{0, \dots, j - 1\}$ , the set  $Y_{i,j}$  computed by the algorithm **wCOC** is defined to be the minimum-weight subset of  $V_{i,j}$  for which the weight of the subgraph  $G[V_{i,j}] - Y_{i,j}$  is at most  $\ell$ . Also recall that once the set  $Y_{i,j}$  is computed, the value of  $\text{dp}[j]$  is updated as follows:

$$\text{dp}[j] = \min \begin{cases} \text{dp}[j] \\ \text{dp}[i] + w(Y_{i,j}) + w(S_j \setminus S_i) \end{cases}$$

When solving the unweighted variant of the problem, we can decrease the weight (i.e., order) of the subgraph  $G[V_{i,j}]$  to at most  $\ell$  by simply deleting  $|V_{i,j}| - \ell$  vertices from  $V_{i,j}$  in a greedy manner. In other words, it is no longer important to decide *which* vertices to delete from  $V_{i,j}$ , but only *how many* vertices to delete. This means that we can replace the entire body of the inner loop by the following line:

$$\text{dp}[j] = \min \begin{cases} \text{dp}[j] \\ \text{dp}[i] + (|V_{i,j}| - \ell) + |S_j \setminus S_i| \end{cases}$$



Hence it suffices to argue that we can precompute the values  $|V_{i,j}|$  and  $|S_j \setminus S_i|$  for every  $j \in \{1, \dots, t\}$  and  $i \in \{0, \dots, j - 1\}$  in  $O(n^2)$  time in total.

Recall that  $V_{i,j} = V(G[\bigcup_{p=i+1}^j K_p \setminus (S_i \cup S_j)])$  by definition, so

$$|V_{i,j}| = \left| \bigcup_{p=i+1}^j K_p \right| - |S_i| - |S_j| + |S_i \cap S_j|.$$

Moreover, it is clear that

$$|S_j \setminus S_i| = |S_j| - |S_i \cap S_j|.$$

The algorithm uCOC starts by computing the sets  $K_0, \dots, K_{t+1}$  and  $S_0, \dots, S_t$  as before in  $O(n^2)$  time, as well as the cardinalities of these sets. For each  $v \in V(G)$ , let  $L(v)$  denote the largest index  $i$  such that  $v \in K_i$ . Observe that we can compute the value  $L(v)$  for all  $v \in V(G)$  in  $O(n^2)$  time in total. The algorithm then computes the value  $|\bigcup_{p=0}^i K_p|$  for every  $i \in \{0, \dots, t\}$ . Using these values, it then computes the value

$$\left| \bigcup_{p=i+1}^j K_p \right| = \left| \bigcup_{q=0}^j K_q \right| - \left| \bigcup_{r=0}^i K_r \right| + |K_i \cap K_{i+1}| = \left| \bigcup_{q=0}^j K_q \right| - \left| \bigcup_{r=0}^i K_r \right| + |S_i|$$

for every  $j \in \{1, \dots, t\}$  and every  $i \in \{0, \dots, j - 1\}$ . Observe that this can also be done in  $O(n^2)$  time in total since all the terms in the expression has been precomputed.

It remains to show that we can compute the value  $|S_i \cap S_j|$  for all indices  $i$  and  $j$  with  $0 \leq i < j \leq t$  in  $O(n^2)$  time in total. Let us fix an index  $i \in \{0, \dots, t\}$ . Since we precomputed the  $L$ -value of each vertex and we can order the vertices in  $S_i$  by increasing  $L$ -value in  $O(n)$  time, we can compute the value  $|S_i \cap S_j| = |\{v \in S_i \mid L(v) \geq j + 1\}| = |S_i \cap S_{j-1}| - |\{v \in S_i \mid L(v) = j + 1\}|$  for all  $j \in \{i + 1, \dots, t\}$ . Observe that the expression  $|\{v \in S_i \mid L(v) = j + 1\}|$  can be computed for every  $j$  by one sweep through  $S_i$  since  $S_i$  is ordered by  $L$ -values. Hence the computation of  $|S_i \cap S_j|$ , for a fixed  $i$  and every  $j$  can be performed in  $O(n)$  time. This completes the proof. □

To conclude this section, we investigate the parameterized complexity and kernelization complexity of COC and wCOC. As mentioned in the introduction, both problems are para-NP-hard when parameterized by  $\ell$  due to the fact that COMPONENT ORDER CONNECTIVITY is equivalent to VERTEX COVER when  $\ell = 1$ . Our next result shows that when restricted to split graphs, both problems are  $W[1]$ -hard when parameterized by  $k$  or by  $\ell$ .

**Theorem 11.** (★) COMPONENT ORDER CONNECTIVITY is  $W[1]$ -hard on split graphs when parameterized by  $k$  or by  $\ell$ .

On the positive side, our next result shows that both problems become fixed-parameter tractable when parameterized by  $k + \ell$ .

**Theorem 12.** WEIGHTED COMPONENT ORDER CONNECTIVITY *can be solved in time  $O(\ell^k(k + \ell)n) = 2^{O(k \log \ell)}n$ .*

*Proof.* Let  $(G, w, k, \ell)$  be an instance of WEIGHTED COMPONENT ORDER CONNECTIVITY, and let  $n = |V(G)|$  and  $m = |E(G)|$ . We assume that every vertex in  $G$  has weight at least 1, as vertices of weight 0 can simply be deleted from the graph. Suppose that  $(G, w, k, \ell)$  is a yes-instance. Then there exists a set  $X \subseteq V(G)$  such that  $w(X) \leq k$  and  $w_{cc}(G - X) \leq \ell$ . Let  $G_1, \dots, G_r$  be the components of  $G - X$ . We can construct a path decomposition of  $G$  by taking as bags the sets  $X \cup V(G_i)$  for all  $i \in \{1, \dots, r\}$ . Since every vertex has weight at least 1, we know that each bag contains at most  $k + \ell$  vertices, implying that  $G$  has treewidth at most  $k + \ell - 1$ . Consequently,  $G$  has at most  $(k + \ell - 1)n$  edges [4]. We may therefore assume that  $m \leq (k + \ell - 1)n$ , as our algorithm can safely reject the instance otherwise.

We now describe a simple branching algorithm that solves the problem. Now, at each step of the algorithm, we use a depth-first search to find a set  $L \subseteq V(G)$  of at most  $\ell + 1$  vertices such that  $w_{cc}(G[L]) \geq \ell + 1$  and  $G[L]$  induces a connected subgraph. If such a set does not exist, then every component of the graph has weight at most  $\ell$ , so we are done. Otherwise, we know that any solution contains a vertex of  $L$ . We therefore branch into  $|L| \leq \ell + 1$  subproblems: for every  $v \in L$ , we create the instance  $(G - v, w, k - w(v), \ell)$ , where we discard the instance in case  $k - w(v) < 0$ . Since the parameter  $k$  decreases by at least 1 at each branching step, the corresponding search tree  $T$  has depth at most  $k$ . Since  $T$  is an  $(\ell + 1)$ -ary tree of depth at most  $k$ , it has at most  $((\ell + 1)^{k+1} - 1)/((\ell + 1) - 1) = O(\ell^k)$  nodes. Due to the assumption that  $m \leq (k + \ell - 1)n$ , the depth-first search at each step can be performed in time  $O(n + m) = O((k + \ell)n)$ . This yields an overall running time of  $O(\ell^k(k + \ell)n) = 2^{O(k \log \ell)}n$ .  $\square$

We now show that the branching algorithm in Theorem 12 is in some sense tight.

**Theorem 13.** (★) *There is no  $2^{o(k \log \ell)}n^{O(1)}$  time algorithm for COMPONENT ORDER CONNECTIVITY, even when restricted to split graphs, unless the ETH fails.*

We conclude this section by showing that WEIGHTED COMPONENT ORDER CONNECTIVITY admits a polynomial kernel.

**Theorem 14.** (★) WEIGHTED COMPONENT ORDER CONNECTIVITY *admits a kernel with at most  $k\ell(k + \ell) + k$  vertices, where each vertex has weight at most  $k + \ell$ .*

## 4 Concluding Remarks

Our NP-completeness result for VERTEX INTEGRITY on co-comparability graphs, together with the polynomial-time algorithm for WEIGHTED VERTEX INTEGRITY by Ray et al. [15] on interval graphs, raises the question whether WEIGHTED

VERTEX INTEGRITY can be solved in polynomial time on permutation graphs. Recall that the unweighted version of this problem can be solved in  $O(n^5)$  time on this graph class [12].

We showed that the COMPONENT ORDER CONNECTIVITY problem does not admit a  $2^{o(k \log \ell)} n^{O(1)}$  time algorithm, unless the ETH fails. Can the problem be solved in time  $c^{k+\ell} n^{O(1)}$  for some constant  $c$ ? Similarly, it would be interesting to investigate whether it is possible to solve VERTEX INTEGRITY in time  $c^p n^{O(1)}$  for some constant  $c$ .

**Acknowledgments.** The authors are grateful to Daniel Lokshtanov for pointing out that  $k \times k$  CLIQUE admits no  $O(2^{k \log k})$  time algorithm unless ETH fails.

## References

1. Bagga, K.S., Beineke, L.W., Goddard, W., Lipman, M.J., Pippert, R.E.: A survey of integrity. *Discr. Appl. Math.* **37**, 13–28 (1992)
2. Barefoot, C.A., Entringer, R., Swart, H.: Vulnerability in graphs—a comparative survey. *J. Combin. Math. Combin. Comput.* **1**(38), 13–22 (1987)
3. Ben-Ameur, W., Mohamed-Sidi, M.-A., Neto, J.: The  $k$ -separator problem. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 337–348. Springer, Heidelberg (2013)
4. Bodlaender, H.L., Fomin, F.V.: Equitable colorings of bounded treewidth graphs. *Theor. Comput. Sci.* **349**(1), 22–30 (2005)
5. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.* **13**(3), 335–379 (1976)
6. Clark, L.H., Entringer, R.C., Fellows, M.R.: Computational complexity of integrity. *J. Combin. Math. Combin. Comput.* **2**, 179–191 (1987)
7. Drange, P.G., Dregi, M.S., van 't Hof, P.: On the Computational Complexity of Vertex Integrity. CoRR, abs/1403.6331 (2014)
8. Fellows, M., Stueckle, S.: The immersion order, forbidden subgraphs and the complexity of network integrity. *J. Combin. Math. Combin. Comput.* **6**, 23–32 (1989)
9. Garey, M.R., Johnson, D.S.: *Computers and Intractability*. W. H. Freeman, New York (1979)
10. Gross, D., Heinig, M., Iswara, L., Kazmierczak, L.W., Luttrell, K., Saccoman, J.T., Suffel, C.: A survey of component order connectivity models of graph theoretic networks. *WSEAS Trans. Math.* **12**, 895–910 (2013)
11. Ho, C.W., Lee, R.C.T.: Counting clique trees and computing perfect elimination schemes in parallel. *Inf. Process. Lett.* **31**(2), 61–68 (1989)
12. Kratsch, D., Kloks, T., Müller, H.: Measuring the vulnerability for classes of intersection graphs. *Discr. Appl. Math.* **77**(3), 259–270 (1997)
13. Li, Y., Zhang, S., Zhang, Q.: Vulnerability parameters of split graphs. *Int. J. Comput. Math.* **85**(1), 19–23 (2008)
14. Ray, S., Deogun, J.S.: Computational complexity of weighted integrity. *J. Combin. Math. Combin. Comput.* **16**, 65–73 (1994)
15. Ray, S., Kannan, R., Zhang, D., Jiang, H.: The weighted integrity problem is polynomial for interval graphs. *Ars Comb.* **79**, 77–95 (2006)
16. Woeginger, G.J.: The toughness of split graphs. *Discr. Math.* **190**(1–3), 295–297 (1998)

# Co-Clustering Under the Maximum Norm

Laurent Bulteau, Vincent Froese<sup>(✉)</sup>, Sepp Hartung, and Rolf Niedermeier

Institut für Softwaretechnik und Theoretische Informatik,  
TU Berlin, Berlin, Germany  
l.bulteau@campus.tu-berlin.de,  
{vincent.froese,sepp.hartung,rolf.niedermeier}@tu-berlin.de

**Abstract.** Co-clustering, that is, partitioning a matrix into “homogeneous” submatrices, has many applications ranging from bioinformatics to election analysis. Many interesting variants of co-clustering are NP-hard. We focus on the basic variant of co-clustering where the homogeneity of a submatrix is defined in terms of minimizing the maximum distance between two entries. In this context, we spot several NP-hard as well as a number of relevant polynomial-time solvable special cases, thus charting the border of tractability for this challenging data clustering problem. For instance, we provide polynomial-time solvability when having to partition the rows and columns into two subsets each (meaning that one obtains four submatrices). When partitioning rows and columns into three subsets each, however, we encounter NP-hardness even for input matrices containing only values from  $\{0, 1, 2\}$ .

## 1 Introduction

Co-clustering, also known as *biclustering*, performs a simultaneous clustering of the rows and columns of a data matrix. Roughly speaking, the problem is, given a numerical input matrix  $A$ , to partition the rows and columns of  $A$  into subsets minimizing a given *cost* function (measuring “homogeneity”). For a given subset of rows  $I$  and a subset of columns  $J$ , the corresponding *cluster* consists of all entries  $a_{ij}$  with  $i \in I$  and  $j \in J$ . The cost function usually defines homogeneity in terms of distances (measured in some norm) between the entries of each cluster. Note that the variant where clusters are allowed to “overlap”, meaning that some rows and columns are contained in multiple clusters, has also been studied [10]. We focus on the non-overlapping variant which can be stated as follows.

### CO-CLUSTERING $_{\mathcal{L}}$

**Input:** A matrix  $A \in \mathbb{R}^{m \times n}$  and two positive integers  $k, \ell \in \mathbb{N}$ .

**Task:** Find a partition of  $A$ 's rows into  $k$  subsets and a partition of  $A$ 's columns into  $\ell$  subsets such that a given cost function (defined with respect to some norm  $\mathcal{L}$ ) is minimized for the corresponding clustering.

---

Laurent Bulteau: Supported by the Alexander von Humboldt Foundation, Bonn, Germany.

Vincent Froese: Supported by the DFG project DAMM (NI 369/13).

Co-clustering is a fundamental paradigm for unsupervised data analysis. Its applications range from microarrays and bioinformatics over recommender systems to election analysis [1, 3, 10]. Due to its enormous practical significance, there is a vast amount of literature discussing various variants; however, due to the observed NP-hardness of “almost all interesting variants” [10], most of the literature deals with heuristic, typically empirically validated algorithms. Indeed, there has been very active research on co-clustering in terms of heuristic algorithms while there is little substantial theoretical work for this important clustering problem. Motivated by an effort towards a deeper theoretical analysis as started by Anagnostopoulos et al. [1], we further refine and strengthen the theoretical investigations on the computational complexity of a natural special case of CO-CLUSTERING $_{\mathcal{L}}$  for the maximum norm  $\mathcal{L} = L_{\infty}$ .

Anagnostopoulos et al. [1] provided a thorough analysis of the polynomial-time approximability of CO-CLUSTERING $_{\mathcal{L}}$  (with respect to  $L_p$ -norms), presenting several constant-factor approximation algorithms. While their algorithms are almost straightforward, relying on one-dimensionally clustering first the rows and then the columns, their main contribution lies in the sophisticated mathematical analysis of the corresponding approximation factors. Note that Jegelka et al. [9] further generalized this approach to higher dimensions, then called *tensor clustering*. In this work, we study (efficient) *exact* instead of approximate solvability. To this end, we investigate a more limited scenario, focussing on CO-CLUSTERING $_{\infty}$ , where the problem comes down to minimizing the maximum distance between entries of a cluster. In particular, our exact and combinatorial polynomial-time algorithms exploit structural properties of the input matrix and do not solely depend on one-dimensional approaches.

*Related Work.* Our main point of reference is the work of Anagnostopoulos et al. [1]. Their focus is on polynomial-time approximation algorithms, but they also provide computational hardness results. In particular, they point to challenging open questions concerning the cases  $k = \ell = 2$ ,  $k = 1$ , or binary input matrices. Within our more restricted setting using the maximum norm, we can resolve parts of these questions. The survey of Madeira and Oliveira [10]<sup>1</sup> provides an excellent overview on the many variations of CO-CLUSTERING $_{\mathcal{L}}$ , there called biclustering, and discusses many applications in bioinformatics and beyond. In particular, they also discuss the special case where the goal is to partition into uniform clusters [8] (that is, each cluster has only one entry value). Our studies indeed generalize this very puristic scenario by not demanding completely uniform clusters (which would correspond to clusters with maximum entry difference 0) but allowing some variation between maximum and minimum cluster entries. Finally, Califano et al. [4] aimed at clusterings where in each submatrix the distance between entries within each row and within each column is upper-bounded. Except for the work of Anagnostopoulos et al. [1], all investigations mentioned above are empirical in nature.

---

<sup>1</sup> According to Google Scholar, accessed September 2014, cited more than 1350 times.

**Table 1.** Overview of results for  $(k, \ell)$ -CO-CLUSTERING $_{\infty}$  with respect to various parameter constellations ( $m$ : number of rows,  $|\Sigma|$ : alphabet size,  $k/\ell$ : size of row/column partition,  $c$ : cost), where  $*$  indicates a value being part of the input and  $\otimes$  indicates that the corresponding value(s) is/are the parameter.

$m$	$ \Sigma $	$k$	$\ell$	$c$	Complexity
*	*	*	*	0	P [Observation 1]
*	2	*	*	*	P [Observation 1]
*	*	1	*	*	P [Theorem 4]
*	*	2	2	*	P [Theorem 5]
*	*	2	$\otimes$	1	FPT [Corollary 2]
$\otimes$	*	$\otimes$	$\otimes$	$\otimes$	FPT [Lemma 2]
*	3	3	3	1	NP-h [Theorem 1]
2	*	2	*	2	NP-h [Theorem 2]

*Our Contributions.* In terms of defining “cluster homogeneity”, we focus on minimizing the maximum distance between two entries within a cluster (maximum norm). Table 1 surveys most of our results. Our main conceptual contribution is to provide a seemingly first study on the exact complexity of a natural special case of CO-CLUSTERING $_{\mathcal{L}}$ , thus potentially stimulating a promising field of research. Our main technical contributions are as follows. Concerning the computational intractability results with respect to even strongly restricted cases, we put a lot of effort in finding the “right” problems to reduce from in order to make the reductions as natural and expressive as possible, thus making non-obvious connections to fields such as geometric set covering. Moreover, seemingly for the first time in the context of co-clustering, we demonstrate that the inherent NP-hardness does not stem from the permutation combinatorics behind: the problem remains NP-hard when all clusters must consist of consecutive rows or columns. This is a strong constraint (the search space size is tremendously reduced—basically from  $\ell^n \cdot k^m$  to  $\binom{n}{\ell} \cdot \binom{m}{k}$ ) which directly gives a polynomial-time algorithm for  $k$  and  $\ell$  being constants. Note that in the general case we have NP-hardness for constant  $k$  and  $\ell$ . Concerning the algorithmic results, we developed a novel reduction to SAT solving (instead of the standard reductions to integer linear programming) which may prove beneficial on the theoretical but also on the practical side. Notably, however, as opposed to previous work on approximation algorithms [1, 9], our methods seem to be tailored for the two-dimensional case (co-clustering) and the higher dimensional case (tensor clustering) appears to be out of reach.

Due to the lack of space, several details are deferred to a full version.

## 2 Formal Definitions and Preliminaries

We use standard terminology for matrices. A matrix  $A = (a_{ij}) \in \mathbb{R}^{m \times n}$  consists of  $m$  rows and  $n$  columns where  $a_{ij}$  denotes the entry in row  $i$  and column  $j$ . We define  $[n] := \{1, 2, \dots, n\}$  and  $[i, j] := \{i, i+1, \dots, j\}$  for  $n, i, j \in \mathbb{N}$ . Throughout this paper, we assume that arithmetical operations require constant time.

$$A = \begin{bmatrix} 1 & 3 & 4 & 1 \\ 2 & 2 & 1 & 3 \\ 0 & 4 & 3 & 0 \end{bmatrix} \qquad \begin{array}{c} J_1 \quad J_2 \\ I_1 \left[ \begin{array}{ccc|c} 1 & 4 & 1 & 3 \\ 2 & 1 & 3 & 2 \\ 0 & 3 & 0 & 4 \end{array} \right] \\ I_2 \end{array} \qquad \begin{array}{c} J_1 \quad J_2 \\ I_1 \left[ \begin{array}{cc|cc} 2 & 3 & 2 & 1 \\ 1 & 1 & 3 & 4 \\ 0 & 0 & 4 & 3 \end{array} \right] \\ I_2 \end{array} \\
 \\
 I_1 = \{1\}, I_2 = \{2, 3\} \qquad I_1 = \{2\}, I_2 = \{1, 3\} \\
 J_1 = \{1, 3, 4\}, J_2 = \{2\} \qquad J_1 = \{1, 4\}, J_2 = \{2, 3\}$$

**Fig. 1.** The example shows two (2, 2)-co-clusterings (middle and right) of the same matrix  $A$  (left-hand side). It demonstrates that by sorting rows and columns according to the co-clustering, the clusters can be illustrated as submatrices of this (permuted) input matrix. The cost of the (2, 2)-co-clustering in the middle is three (because of the two left clusters) and that of the (2, 2)-co-clustering on the right-hand side is one.

*Problem Definition.* We follow the terminology of Anagnostopoulos et al. [1]. For a matrix  $A \in \mathbb{R}^{m \times n}$ , a  $(k, \ell)$ -co-clustering is a pair  $(\mathcal{I}, \mathcal{J})$  consisting of a  $k$ -partition  $\mathcal{I} = \{I_1, \dots, I_k\}$  of the rows of  $A$  (more specifically, a partition of the row indices  $[m]$ ) and an  $\ell$ -partition  $\mathcal{J} = \{J_1, \dots, J_\ell\}$  of the columns (the column indices  $[n]$ , respectively) of  $A$ . We call the elements of  $\mathcal{I}$  ( $\mathcal{J}$ ) row blocks (column blocks, resp.). Additionally, we require  $\mathcal{I}$  and  $\mathcal{J}$  to not contain empty sets. For  $(r, s) \in [k] \times [\ell]$ , the set  $A_{rs} := \{a_{ij} \in A \mid (i, j) \in I_r \times J_s\}$  is called a *cluster*.

The cost of a co-clustering (under maximum norm, which is the only norm we consider here) is defined as the maximum difference between any two entries in any cluster, formally  $\text{cost}(\mathcal{I}, \mathcal{J}) := \max_{(r,s) \in [k] \times [\ell]} (\max A_{rs} - \min A_{rs})$ . Herein,  $\max A_{rs}$  ( $\min A_{rs}$ ) denotes the maximum (minimum, resp.) entry in  $A_{rs}$ .

The decision variant of  $\text{CO-CLUSTERING}_{\mathcal{L}}$  with maximum norm is as follows.

#### CO-CLUSTERING $_{\infty}$

**Input:** A matrix  $A \in \mathbb{R}^{m \times n}$ , integers  $k, \ell \in \mathbb{N}$ , and a cost  $c \geq 0$ .

**Question:** Is there a  $(k, \ell)$ -co-clustering  $(\mathcal{I}, \mathcal{J})$  of  $A$  with  $\text{cost}(\mathcal{I}, \mathcal{J}) \leq c$ ?

See [Figure 1](#) for an introductory example. We define  $\Sigma := \{a_{ij} \in A \mid (i, j) \in [m] \times [n]\}$  to be the *alphabet* of the input matrix  $A$  (consisting of the values that occur in  $A$ ). We use the abbreviation  $(k, \ell)$ -CO-CLUSTERING $_{\infty}$  to refer to CO-CLUSTERING $_{\infty}$  with constants  $k, \ell \in \mathbb{N}$ , and by  $(k, *)$ -CO-CLUSTERING $_{\infty}$  we refer to the case where only  $k$  is constant and  $\ell$  is part of the input. Clearly, CO-CLUSTERING $_{\infty}$  is symmetric with respect to  $k$  and  $\ell$  in the sense that any  $(k, \ell)$ -co-clustering of a matrix  $A$  is equivalent to an  $(\ell, k)$ -co-clustering of the transposed matrix  $A^T$ . Hence, we always assume that  $k \leq \ell$ .

We next collect some simple observations. First, determining whether there is a cost-zero (perfect) co-clustering is easy. Moreover, since, for a binary alphabet, the only interesting case is a perfect co-clustering, we get the following.

**Observation 1.** CO-CLUSTERING $_{\infty}$  is solvable in polynomial time for cost zero and also for any size-two alphabet.

*Proof.* Let  $(A, k, \ell, 0)$  be a  $\text{CO-CLUSTERING}_\infty$  instance. For a  $(k, \ell)$ -co-clustering with cost 0, it holds that all entries of a cluster are equal. This is only possible if there are at most  $k$  different rows and at most  $\ell$  different columns in  $A$  since otherwise there will be a cluster containing two different entries. Thus, the case  $c = 0$  can be solved by lexicographically sorting the rows and columns of  $A$ .  $\square$

We further observe that the input matrix can, without loss of generality, be assumed to contain only integer values (by some rescaling arguments preserving the distance relations between elements).

**Observation 2.** *For any  $\text{CO-CLUSTERING}_\infty$ -instance with arbitrary alphabet  $\Sigma \subseteq \mathbb{R}$ , one can find in  $O((mn)^2)$  time an equivalent instance with alphabet  $\Sigma' \subseteq \mathbb{Z}$  and cost value  $c' \in \mathbb{N}$ .*

*Parameterized Algorithmics.* We briefly introduce the relevant notions from parameterized algorithmics. A parameterized problem, each instance consisting of the “classical” problem instance  $I$  and an integer  $k$ , is *fixed-parameter tractable* (FPT) if there is a computable function  $f$  and an algorithm solving any instance in  $f(k) \cdot |I|^{O(1)}$  time. The corresponding algorithm is called FPT-algorithm.

### 3 Intractability Results

In the previous section, we observed that  $\text{CO-CLUSTERING}_\infty$  is easy to solve for binary input matrices (Observation 1). In contrast to this, we show in this section that its computational complexity significantly changes as soon as the input matrix contains at least three different entries. In fact, even for very restricted special cases we can show NP-hardness. These special cases comprise co-clusterings with a constant number of clusters or input matrices with only two rows. We also show NP-hardness of finding co-clusterings where the row and column partitions are only allowed to contain consecutive blocks.

#### 3.1 Constant Number of Clusters

We start by showing that for input matrices containing three different entries,  $\text{CO-CLUSTERING}_\infty$  is NP-hard even if the co-clustering consists only of nine clusters.

**Theorem 1.**  *$(3, 3)$ - $\text{CO-CLUSTERING}_\infty$  is NP-hard for  $\Sigma = \{0, 1, 2\}$ .*

*Proof.* We reduce from the NP-complete 3-COLORING [7], where the task is to partition the vertex set of an undirected graph into three independent sets. Let  $G = (V, E)$  be a 3-COLORING instance with  $V = \{v_1, \dots, v_n\}$  and  $E = \{e_1, \dots, e_m\}$ . We construct a  $(3, 3)$ - $\text{CO-CLUSTERING}_\infty$  instance  $(A \in \{0, 1, 2\}^{m \times n}, k := 3, \ell := 3, c := 1)$  as follows. The columns of  $A$  correspond to the vertices  $V$  and the rows



correspond to the edges  $E$ . For an edge  $e_i = (v_j, v_{j'}) \in E$  with  $j < j'$ , we set  $a_{ij} := 0$  and  $a_{ij'} := 2$ . All other matrix entries are set to one. Hence, each row corresponding to an edge  $\{v_j, v_{j'}\}$  consists of 1-entries except for the columns  $j$  and  $j'$ , which contain 0 and 2. Thus, every co-clustering of  $A$  with cost at most  $c = 1$  puts column  $j$  and column  $j'$  into different column blocks. We next prove that there is a  $(3, 3)$ -co-clustering of  $A$  with cost at most  $c = 1$  if and only if  $G$  admits a 3-coloring.

First, assume that  $V_1, V_2, V_3$  is a partition of the vertex set  $V$  into three independent sets. We define a  $(3, 3)$ -co-clustering  $(\mathcal{I}, \mathcal{J})$  of  $A$  as follows. The column partition  $\mathcal{J} := \{J_1, J_2, J_3\}$  one-to-one corresponds to the three sets  $V_1, V_2, V_3$ , that is,  $J_s := \{i \mid v_i \in V_s\}$  for all  $s \in \{1, 2, 3\}$ . By the construction above, each row has exactly two non-1 entries being 0 and 2. We define the type of a row to be a permutation of 0, 1, 2, denoting which of the column blocks  $J_1, J_2, J_3$  contain the 0-entry and the 2-entry. For example, a row is of type  $(2, 0, 1)$  if it has a 2 in a column of  $J_1$  and a 0 in a column of  $J_2$ . The row partition  $\mathcal{I} := \{I_1, I_2, I_3\}$  is defined as follows: All rows of type  $(0, 2, 1)$  or  $(0, 1, 2)$  are put into  $I_1$ . Rows of type  $(2, 0, 1)$  or  $(1, 0, 2)$  are contained in  $I_2$  and the remaining rows of type  $(2, 1, 0)$  or  $(1, 2, 0)$  are contained in  $I_3$ . Clearly, for  $(\mathcal{I}, \mathcal{J})$ , it holds that the non-1 entries in any cluster are either all 0 or all 2, implying that  $\text{cost}(\mathcal{I}, \mathcal{J}) \leq 1$ .

Next, assume that  $(\mathcal{I}, \{J_1, J_2, J_3\})$  is a  $(3, 3)$ -co-clustering of  $A$  with cost at most 1. The vertex sets  $V_1, V_2, V_3$ , where  $V_s$  contains the vertices corresponding to the columns in  $J_s$ , form three independent sets: If an edge connects two vertices in  $V_s$ , then the corresponding row would have the 0-entry and the 2-entry in the same column block  $J_s$ , yielding a cost of 2, which is a contradiction.  $\square$

**Theorem 1** can even be strengthened further.

**Corollary 1.**  $\text{CO-CLUSTERING}_\infty$  is NP-hard even when  $k = m$  (that is, each row is in its own cluster),  $\ell$  is fixed with  $\ell \geq 3$ ,  $\Sigma = \{0, 1, 2\}$ , and the column blocks are forced to have equal sizes  $|J_1| = \dots = |J_\ell|$ .

*Proof (Sketch).* Note that the reduction in **Theorem 1** can easily be adapted to the NP-hard  $\ell$ -COLORING problem with balanced partition sizes [7]. Note also that the proof holds for any  $k \geq 3$ . Hence, the problem is NP-hard for  $k = m$  row blocks.  $\square$

### 3.2 Constant Number of Rows

The reduction in the proof of **Theorem 1** outputs matrices with an unbounded number of rows and columns containing only three different values. We now show that also the “dual restriction” is NP-hard, that is, the input matrix only has a constant number of rows (two) but contains an unbounded number of different values. Interestingly, this special case is closely related to a two-dimensional variant of geometric set covering.

**Theorem 2.**  $\text{CO-CLUSTERING}_\infty$  is NP-hard for  $k = m = 2$ .

*Proof.* We give a polynomial-time reduction from the NP-complete BOX COVER problem [6]. Given a set  $P \subseteq \mathbb{Z}^2$  of  $n$  points in the plane and  $\ell \in \mathbb{N}$ , BOX COVER is the problem to decide whether there are  $\ell$  squares  $S_1, \dots, S_\ell$ , each with side length 2, covering  $P$ , that is,  $P \subseteq \bigcup_{1 \leq i \leq \ell} S_i$ .

Let  $I = (P, \ell)$  be a BOX COVER instance. We define the instance  $I' := (A, k, \ell', c)$  as follows: The matrix  $A \in \mathbb{Z}^{2 \times n}$  has the points  $p_1, \dots, p_n$  in  $P$  as columns. Further, we set  $k := 2$ ,  $\ell' := \ell$ ,  $c := 2$ .

The correctness can be seen as follows: Assume that  $I$  is a yes-instance, that is, there are  $\ell$  squares  $S_1, \dots, S_\ell$  covering all points in  $P$ . We define  $J_1 := \{i \mid p_i \in P \cap S_1\}$  and  $J_j := \{i \mid p_i \in P \cap S_j \setminus (\bigcup_{1 \leq l < j} S_l)\}$  for all  $2 \leq j \leq \ell$ . Note that  $(\mathcal{I} = \{\{1\}, \{2\}\}, \mathcal{J} = \{J_1, \dots, J_\ell\})$  is a  $(2, \ell)$ -co-clustering of  $A$ . Moreover, since all points with indices in  $J_j$  lie inside a square with side length 2, it holds that each pair of entries in  $A_{1j}$  as well as in  $A_{2j}$  has distance at most 2, implying  $\text{cost}(\mathcal{I}, \mathcal{J}) \leq 2$ .

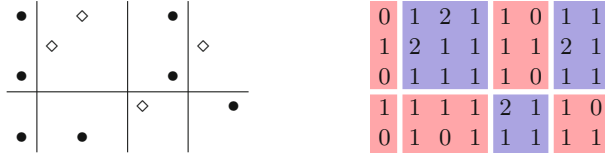
Conversely, if  $I'$  is a yes-instance, then let  $(\{\{1\}, \{2\}\}, \mathcal{J})$  be the  $(2, \ell)$ -co-clustering of cost at most 2. For any  $J_i \in \mathcal{J}$ , it holds that all points corresponding to the columns in  $J_i$  have pairwise distance at most 2 in both coordinates. Thus, there exists a square of side length 2 covering all of them.  $\square$

### 3.3 Clustering into Consecutive Clusters

One is tempted to assume that the hardness of the previous special cases of CO-CLUSTERING $_\infty$  is rooted in the fact that we are allowed to choose arbitrary subsets for the corresponding row and column partitions since the problem remains hard even for a constant number of clusters and also with equal cluster sizes. Hence, in this section, we consider a restricted version of CO-CLUSTERING $_\infty$ , where the row and the column partition has to consist of consecutive blocks. Formally, for row indices  $R = \{r_1, \dots, r_{k-1}\}$  with  $1 < r_1 < \dots < r_{k-1} \leq m$  and column indices  $C = \{c_1, \dots, c_{\ell-1}\}$  with  $1 < c_1 < \dots < c_{\ell-1} \leq n$ , the corresponding *consecutive*  $(k, \ell)$ -co-clustering  $(\mathcal{I}_R, \mathcal{J}_C)$  is defined as

$$\begin{aligned} \mathcal{I}_R &:= \{\{1, \dots, r_1 - 1\}, \{r_1, \dots, r_2 - 1\}, \dots, \{r_{k-1}, \dots, m\}\}, \\ \mathcal{J}_C &:= \{\{1, \dots, c_1 - 1\}, \{c_1, \dots, c_2 - 1\}, \dots, \{c_{\ell-1}, \dots, n\}\}. \end{aligned}$$

The CONSECUTIVE CO-CLUSTERING $_\infty$  problem now is to find a consecutive  $(k, \ell)$ -co-clustering of a given input matrix with a given cost. Again, also this restriction is not sufficient to overcome the inherent intractability of co-clustering, that is, we prove it to be NP-hard. Similarly to Section 3.2, we encounter a close relation of consecutive co-clustering to a geometric problem, namely to find an optimal discretization of the plane [5]. The NP-hard OPTIMAL DISCRETIZATION problem [5] is the following: Given a set  $S$  of points in the plane, each either colored black  $B$  or white  $W$ , and integers  $k, \ell \in \mathbb{N}$ , decide whether there is a consistent set of  $k$  horizontal and  $\ell$  vertical (axis-parallel) lines. That is, the vertical and horizontal lines partition the plane into rectangular regions such that no region contains two points of different colors (see Figure 2 for an example). Here, a vertical (horizontal) line is a simple number denoting its  $x$ -( $y$ -)coordinate.



**Fig. 2.** Example instance of OPTIMAL DISCRETIZATION (left) and the corresponding instance of CONSECUTIVE CO-CLUSTERING<sub>∞</sub> (right). A solution to the CONSECUTIVE CO-CLUSTERING<sub>∞</sub> instance (shaded clusters) naturally translates into a consistent set of lines.

**Theorem 3.** CONSECUTIVE CO-CLUSTERING<sub>∞</sub> is NP-hard for  $\Sigma = \{0, 1, 2\}$ .

*Proof (Sketch).* We give a polynomial-time reduction from OPTIMAL DISCRETIZATION. Let  $(S, k, \ell)$  be an OPTIMAL DISCRETIZATION instance and let  $X := \{x_1^*, \dots, x_n^*\}$  be the set of different  $x$ -coordinates and let  $Y := \{y_1^*, \dots, y_m^*\}$  be the set of different  $y$ -coordinates of the points in  $S$ . Note that  $n$  and  $m$  can be smaller than  $|S|$  since two points can have the same  $x$ - or  $y$ -coordinate. Furthermore, assume that  $x_1^* < \dots < x_n^*$  and  $y_1^* < \dots < y_m^*$ . We now define the CONSECUTIVE CO-CLUSTERING<sub>∞</sub> instance  $(A, k + 1, \ell + 1, c)$  as follows: The matrix  $A \in \{0, 1, 2\}^{m \times n}$  has columns labeled with  $x_1^*, \dots, x_n^*$  and rows labeled with  $y_1^*, \dots, y_m^*$ . For  $(x, y) \in X \times Y$ , the entry  $a_{xy}$  is defined as 0 if  $(x, y) \in B$ , 2 if  $(x, y) \in W$ , and otherwise 1. The cost is set to  $c := 1$ . Clearly, this instance can be constructed in polynomial time. We prove correctness in a full version of the paper.  $\square$

Note that even though CONSECUTIVE CO-CLUSTERING<sub>∞</sub> is NP-hard, there still is some difference in its computational complexity compared to the general version. In contrast to CO-CLUSTERING<sub>∞</sub>, the consecutive version is polynomial-time solvable for constants  $k$  and  $\ell$  by trying out all  $O(m^k n^\ell)$  consecutive partitions of the rows and columns.

## 4 Tractability Results

In Section 3, we showed that CO-CLUSTERING<sub>∞</sub> is NP-hard for  $k = \ell = 3$  and also for  $k = 2$  in case of unbounded  $\ell$  and  $|\Sigma|$ . In contrast to these hardness results, we now investigate which parameter combinations yield tractable cases. It turns out that the problem is polynomial-time solvable for  $k = \ell = 2$  and for  $k = 1$ . We can even solve the case  $k = 2$  and  $\ell \geq 3$  for  $|\Sigma| = 3$  in polynomial time by showing that this case is in fact equivalent to the case  $k = \ell = 2$ . Note that these tractability results nicely complement the hardness results from Section 3. We further show fixed-parameter tractability for the parameters size of the alphabet  $|\Sigma|$  and the number of column blocks  $\ell$ .

We start by describing a reduction of CO-CLUSTERING<sub>∞</sub> to CNF-SAT (the satisfiability problem for boolean formulas in conjunctive normal form). Later on, it will be used in some special cases (see Theorem 5 and Theorem 7) because there the corresponding formula—or an equivalent formula—only consists of

clauses containing two literals, thus being a polynomial-time solvable 2-SAT-instance.

### 4.1 Reduction to CNF-SAT Solving

To start with, we introduce the concept of cluster boundaries, which are basically lower and upper bounds for the values in a cluster of a co-clustering. Formally, given two integers  $k, \ell$ , an alphabet  $\Sigma$ , and a cost  $c$ , we define a *cluster boundary* to be a matrix  $\mathcal{U} = (u_{rs}) \in \Sigma^{k \times \ell}$ . We say that a  $(k, \ell)$ -co-clustering of  $A$  *satisfies* a cluster boundary  $\mathcal{U}$  if  $A_{rs} \subseteq [u_{rs}, u_{rs} + c]$  for all  $(r, s) \in [k] \times [\ell]$ . It can easily be seen that a given  $(k, \ell)$ -co-clustering has cost at most  $c$  if and only if it satisfies at least one cluster boundary  $(u_{rs})$ , namely, the one with  $u_{rs} = \min A_{rs}$ .

The following “subtask” of  $\text{CO-CLUSTERING}_\infty$  can be reduced to a certain CNF-SAT instance: Given a cluster boundary  $\mathcal{U}$  and a  $\text{CO-CLUSTERING}_\infty$ -instance  $I$ , find a co-clustering for  $I$  that satisfies  $\mathcal{U}$ . The reduction provided by the following lemma can be used to obtain exact  $\text{CO-CLUSTERING}_\infty$  solutions with the help of SAT solvers and we use it in our subsequent algorithms.

**Lemma 1.** *Given a  $\text{CO-CLUSTERING}_\infty$ -instance  $(A, k, \ell, c)$  and a cluster boundary  $\mathcal{U}$ , one can construct in polynomial time a CNF-SAT-instance  $\phi$  with at most  $\max\{k, \ell\}$  variables per clause such that  $\phi$  is satisfiable if and only if there is a  $(k, \ell)$ -co-clustering of  $A$  which satisfies  $\mathcal{U}$ .*

*Proof.* Given an instance  $(A, k, \ell, c)$  of  $\text{CO-CLUSTERING}_\infty$  and a cluster boundary  $\mathcal{U} = (u_{rs}) \in \Sigma^{k \times \ell}$ , we define the following boolean variables: For each  $(i, r) \in [m] \times [k]$ , the variable  $x_{i,r}$  represents the expression “row  $i$  could be put into row block  $I_r$ ”. Similarly, for each  $(j, s) \in [n] \times [\ell]$ , the variable  $y_{j,s}$  represents that “column  $j$  could be put into column block  $J_s$ ”.

We now define a boolean CNF formula  $\phi_{A,\mathcal{U}}$  containing the following clauses: A clause  $R_i := x_{i,1} \vee x_{i,2} \vee \dots \vee x_{i,k}$  for each row  $i \in [m]$  and a clause  $C_j := y_{j,1} \vee y_{j,2} \vee \dots \vee y_{j,\ell}$  for each column  $j \in [n]$ . Additionally, for each  $(i, j) \in [m] \times [n]$  and each  $(r, s) \in [k] \times [\ell]$  such that element  $a_{ij}$  does not fit into the cluster boundary at coordinate  $(r, s)$ , that is,  $a_{ij} \notin [u_{rs}, u_{rs} + c]$ , there is a clause  $B_{ijrs} := \neg x_{i,r} \vee \neg y_{j,s}$ . Note that the clauses  $R_i$  and  $C_j$  ensure that row  $i$  and column  $j$  are put into some row and some column block respectively. The clause  $B_{ijrs}$  expresses that it is impossible to have both row  $i$  in block  $I_r$  and column  $j$  in block  $J_s$  if  $a_{ij}$  does not satisfy  $u_{rs} \leq a_{ij} \leq u_{rs} + c$ . Clearly,  $\phi_{A,\mathcal{U}}$  is satisfiable if and only if there exists a  $(k, \ell)$ -co-clustering of  $A$  satisfying the cluster boundary  $\mathcal{U}$ . Note that  $\phi_{A,\mathcal{U}}$  consists of  $O(km + \ell n)$  variables and  $O(mnk\ell)$  clauses.  $\square$

### 4.2 Polynomial-Time Solvability

We first present a fairly simple algorithm for  $(1, *)$ - $\text{CO-CLUSTERING}_\infty$ , that is, the variant where all rows belong to one row block.

**Theorem 4.**  *$(1, *)$ - $\text{CO-CLUSTERING}_\infty$  is solvable in  $O(n(m + \log n))$  time.*

**Algorithm 1.** Algorithm for  $(1, *)$ -CO-CLUSTERING $_{\infty}$ **Input:**  $A \in \mathbb{R}^{m \times n}$ ,  $\ell \geq 1$ ,  $c \geq 0$ **Output:** A partition of  $[n]$  into at most  $\ell$  blocks yielding a cost of at most  $c$ , or no if no such partition exists.

---

```

1 for  $j \leftarrow 1$  to  $n$  do
2    $\alpha_j \leftarrow \min\{a_{ij} \mid 1 \leq i \leq m\}$ ;
3    $\beta_j \leftarrow \max\{a_{ij} \mid 1 \leq i \leq m\}$ ;
4  $\mathcal{N} \leftarrow [n]$ ;
5 for  $s \leftarrow 1$  to  $\ell$  do
6   Let  $x_s \in \mathcal{N}$  be the index such that  $\alpha_{x_s}$  is minimal;
7    $J_s \leftarrow \{j \in \mathcal{N} \mid \beta_j - \alpha_{x_s} \leq c\}$ ;
8    $\mathcal{N} \leftarrow \mathcal{N} \setminus J_s$ ;
9   if  $\mathcal{N} = \emptyset$  then
10    return  $(J_1, \dots, J_s)$ ;
11 return no;

```

---

*Proof.* We show that Algorithm 1 solves  $(1, *)$ -CO-CLUSTERING $_{\infty}$ . In fact, it even computes the minimum  $\ell$  such that  $A$  has a  $(1, \ell)$ -co-clustering of cost  $c$ .

If Algorithm 1 returns  $(J_1, \dots, J_{\ell'})$  at line 10, then this is a column partition into  $\ell' \leq \ell$  blocks satisfying the cost constraint. First, it is a partition by construction: The sets  $J_s$  are successively removed from  $\mathcal{N}$  until it is empty. Now, let  $s \in [\ell']$ . Then, for all  $j \in J_s$ , it holds  $\alpha_j \geq \alpha_{x_s}$  (by definition of  $x_s$ ) and  $\beta_j \leq \alpha_{x_s} + c$  (by definition of  $J_s$ ). Thus,  $A_{1s} \subseteq [\alpha_{x_s}, \alpha_{x_s} + c]$  holds for all  $s \in [\ell']$ , which yields  $\text{cost}(\{[m]\}, \{J_1, \dots, J_{\ell'}\}) \leq c$ . Otherwise, if Algorithm 1 returns no at Line 11, then it has computed indices  $x_s, s \in [\ell]$ , and there exists at least one element  $x_{\ell+1}$  in  $\mathcal{N}$  when the algorithm terminates. Consider any  $1 \leq s < s' \leq \ell + 1$ . By construction,  $x_{s'} \notin J_s$ . Therefore,  $\beta_{x_{s'}} > \alpha_{x_s} + c$  holds, and columns  $x_s$  and  $x_{s'}$  contain elements with distance more than  $c$ . Thus, in any co-clustering with cost at most  $c$ , columns  $x_1, \dots, x_{\ell+1}$  must be in different blocks, which is impossible by the pigeon-hole principle. Hence, this is indeed a no-instance.

The time complexity is seen as follows. The first loop examines all elements of the matrix in  $O(mn)$  time. The second loop can be performed in  $O(n)$  time if the  $\alpha_j$  and the  $\beta_j$  are sorted beforehand, requiring  $O(n \log n)$  time. Overall, the running time is in  $O(n(m + \log n))$ .  $\square$

From now on, we focus on the  $k = 2$  case, that is, we need to partition the rows into two blocks. We first consider the simplest case, where also  $\ell = 2$ .

**Theorem 5.**  $(2, 2)$ -CO-CLUSTERING $_{\infty}$  is solvable in  $O(|\Sigma|^2 mn)$  time.

*Proof.* We use the reduction to CNF-SAT provided by Lemma 1. First, note that a cluster boundary  $\mathcal{U} \in \Sigma^{2 \times 2}$  can only be satisfied if it contains the elements  $\min \Sigma$  and  $\min\{a \in \Sigma \mid a \geq \max \Sigma - c\}$ . The algorithm enumerates all  $O(|\Sigma|^2)$  of these cluster boundaries. For a fixed  $\mathcal{U}$ , we construct the boolean formula  $\phi_{A, \mathcal{U}}$ . Observe that this formula is in 2-CNF form: The formula consists

of  $k$ -clauses,  $\ell$ -clauses, and 2-clauses, and we have  $k = \ell = 2$ . Hence, we can determine whether it is satisfiable in linear time [2] (note that the size of the formula is in  $O(mn)$ ). Overall, the input is a yes-instance if and only if  $\phi_{A,\mathcal{U}}$  is satisfiable for some cluster boundary  $\mathcal{U}$ .  $\square$

Finally, we claim that it is possible to extend the above result to any number of column blocks for size-three alphabets (refer to the full version for a proof).

**Theorem 6.**  $(2, *)$ -CO-CLUSTERING $_{\infty}$  is polynomial-time solvable for  $|\Sigma| = 3$ .

### 4.3 Fixed-Parameter Tractability

We develop an algorithm solving  $(2, *)$ -CO-CLUSTERING $_{\infty}$  for  $c = 1$  based on our reduction to CNF-SAT (see Lemma 1). The main idea is, given matrix  $A$  and cluster boundary  $\mathcal{U}$ , to simplify the boolean formula  $\phi_{A,\mathcal{U}}$  into a 2-SAT formula which can be solved efficiently. This is made possible by the constraint on the cost, which imposes a very specific structure on the cluster boundary. This approach requires to enumerate all (exponentially many) possible cluster boundaries, but yields fixed-parameter tractability for the combined parameter  $(\ell, |\Sigma|)$ .

**Theorem 7.**  $(2, *)$ -CO-CLUSTERING $_{\infty}$  is  $O(|\Sigma|^{3\ell} n^2 m^2)$ -time solvable for  $c = 1$ .

A subresult in the proof of Theorem 7 (deferred to a full version) is the following lemma, which we use to solve the case where the number  $2^m$  of possible row partitions is less than  $|\Sigma|^{\ell}$ .

**Lemma 2.** For a fixed row partition  $\mathcal{I}$ , one can solve CO-CLUSTERING $_{\infty}$  in  $O(|\Sigma|^{k\ell} mn\ell)$  time. Moreover, CO-CLUSTERING $_{\infty}$  is fixed-parameter tractable with respect to the combined parameter  $(m, k, \ell, c)$ .

*Proof.* Given a fixed row partition  $\mathcal{I}$ , the algorithm enumerates all  $|\Sigma|^{k\ell}$  different cluster boundaries  $\mathcal{U} = (u_{rs})$ . We say that a given column  $j$  fits in column block  $J_s$  if, for each  $r \in [k]$  and  $i \in I_r$ , we have  $a_{ij} \in [u_{rs}, u_{rs} + c]$  (this can be decided in  $O(m)$  time for any pair  $(j, s)$ ). The input is a yes-instance if and only if for some cluster boundary  $\mathcal{U}$ , every column fits in at least one column block.

Fixed-parameter tractability with respect to  $(m, k, \ell, c)$  is obtained from two simple further observations. First, all possible row partitions can be enumerated in  $O(k^m)$  time. Second, since each of the  $k\ell$  clusters contains at most  $c + 1$  different values, the alphabet size  $|\Sigma|$  for yes-instances is upper-bounded by  $(c + 1)k\ell$ .  $\square$

Finally, we obtain the following simple corollary.

**Corollary 2.**  $(2, *)$ -CO-CLUSTERING $_{\infty}$  with  $c = 1$  is fixed-parameter tractable with respect to parameter  $|\Sigma|$  and with respect to parameter  $\ell$ .

*Proof.* Theorem 7 presents an FPT-algorithm with respect to the combined parameter  $|\Sigma|$  and  $\ell$ . For  $(2, *)$ -CO-CLUSTERING $_{\infty}$  with  $c = 1$ , both parameters are equivalent. Indeed,  $\ell < |\Sigma|^2$  (otherwise there are two column blocks with identical cluster boundaries, which could be merged) and  $|\Sigma| < 2(c + 1)\ell = 4\ell$  (each column block may contain two intervals, each covering at most  $c + 1$  elements).  $\square$

## 5 Conclusion

Contrasting previous theoretical work on approximation algorithms [1, 9], we started to closely investigate the time complexity of exactly solving the NP-hard  $\text{CO-CLUSTERING}_\infty$  problem, contributing a detailed view on its computational complexity landscape. Refer to Table 1 for an overview on most of our results. From a practical perspective, both our polynomial-time algorithms and our reduction to CNF-SAT solving—notably, exact solving approaches for  $\text{CO-CLUSTERING}_\mathcal{L}$  so far mostly rely on integer linear programming—may prove useful.

Several open questions derive from our work. Perhaps the most pressing open question is whether the most basic three-dimensional case—(2,2,2)- $\text{CO-CLUSTERING}_\infty$  on three-dimensional input matrices—is polynomial-time solvable or NP-hard. Indeed, other than the techniques for deriving approximation algorithms [1, 9] our exact methods do not seem to generalize to higher dimensions.

**Acknowledgments.** We thank Stéphane Vialette for stimulating discussions.

## References

1. Anagnostopoulos, A., Dasgupta, A., Kumar, R.: A constant-factor approximation algorithm for co-clustering. *Theory of Computing* **8**, 597–622 (2012)
2. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inform. Process. Lett.* **8**(3), 121–123 (1979)
3. Banerjee, A., Dhillon, I.S., Ghosh, J., Merugu, S., Modha, D.S.: A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. *J. Mach. Learn. Res.* **8**, 1919–1986 (2007)
4. Califano, C., Stolovitzky, G., Tu, Y.: Analysis of Gene Expression Microarrays for Phenotype Classification. In: *Proc. Eighth ISMB*, pp. 75–85. AAAI (2000)
5. Chlebus, B.S., Nguyen, S.H.: On finding optimal discretizations for two attributes. In: Polkowski, L., Skowron, A. (eds.) *RSCTC 1998*. LNCS (LNAI), vol. 1424, pp. 537–544. Springer, Heidelberg (1998)
6. Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.* **12**(3), 133–137 (1981)
7. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company (1979)
8. Hartigan, J.A.: Direct clustering of a data matrix. *J. Amer. Stat. Assoc.* **67**(337), 123–129 (1972)
9. Jegelka, S., Sra, S., Banerjee, A.: Approximation algorithms for tensor clustering. In: Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S. (eds.) *ALT 2009*. LNCS, vol. 5809, pp. 368–383. Springer, Heidelberg (2009)
10. Madeira, S.C., Oliveira, A.L.: Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM TCBB* **1**(1), 24–45 (2004)

# **Computational Geometry II**



# The Price of Order

Prosenjit Bose<sup>1</sup>, Pat Morin<sup>1</sup>, and André van Renssen<sup>1,2</sup>(✉)

<sup>1</sup> School of Computer Science, Carleton University, Ottawa, Canada

{jit,morin}@scs.carleton.ca

<sup>2</sup> National Institute of Informatics, Tokyo, Japan

andre@cg.scs.carleton.ca

**Abstract.** We present tight bounds on the spanning ratio of a large family of ordered  $\theta$ -graphs. A  $\theta$ -graph partitions the plane around each vertex into  $m$  disjoint cones, each having aperture  $\theta = 2\pi/m$ . An ordered  $\theta$ -graph is constructed by inserting the vertices one by one and connecting each vertex to the closest previously-inserted vertex in each cone. We show that for any integer  $k \geq 1$ , ordered  $\theta$ -graphs with  $4k + 4$  cones have a tight spanning ratio of  $1 + 2\sin(\theta/2)/(\cos(\theta/2) - \sin(\theta/2))$ . We also show that for any integer  $k \geq 2$ , ordered  $\theta$ -graphs with  $4k + 2$  cones have a tight spanning ratio of  $1/(1 - 2\sin(\theta/2))$ . We provide lower bounds for ordered  $\theta$ -graphs with  $4k + 3$  and  $4k + 5$  cones. For ordered  $\theta$ -graphs with  $4k + 2$  and  $4k + 5$  cones these lower bounds are strictly greater than the worst case spanning ratios of their unordered counterparts. These are the first results showing that ordered  $\theta$ -graphs have worse spanning ratios than unordered  $\theta$ -graphs. Finally, we show that, unlike their unordered counterparts, the ordered  $\theta$ -graphs with 4, 5, and 6 cones are not spanners.

## 1 Introduction

In a weighted graph  $G$ , let the distance  $\delta_G(u, v)$  between two vertices  $u$  and  $v$  be the length of the shortest path between  $u$  and  $v$  in  $G$ . A subgraph  $H$  of  $G$  is a  $t$ -spanner of  $G$  if for all pairs of vertices  $u$  and  $v$ ,  $\delta_H(u, v) \leq t \cdot \delta_G(u, v)$ ,  $t \geq 1$ . The *spanning ratio* of  $H$  is the smallest  $t$  for which  $H$  is a  $t$ -spanner. The graph  $G$  is referred to as the *underlying graph* [13]. We consider the situation where the underlying graph  $G$  is a straightline embedding of the complete graph on a set of  $n$  points in the plane. The weight of each edge  $uv$  is the Euclidean distance  $|uv|$  between  $u$  and  $v$ . A spanner of such a graph is called a *geometric spanner*. We look at a specific type of geometric spanner:  $\theta$ -graphs.

Introduced independently by Clarkson [10] and Keil [12],  $\theta$ -graphs are constructed as follows: for each vertex  $u$ , we partition the plane into  $m$  disjoint cones with apex  $u$ , each having aperture  $\theta = 2\pi/m$ . The  $\theta$ -graph is constructed by, for each cone with apex  $u$ , connecting  $u$  to the vertex  $v$  whose projection along the bisector of the cone is closest. When  $m$  cones are used, we denote the resulting

---

Research supported in part by NSERC and Carleton University's President's 2010 Doctoral Fellowship.

André van Renssen is also part of the JST, ERATO Kawarabayashi Large Graph Project.

$\theta$ -graph as  $\theta_m$ . Ruppert and Seidel [14] showed that the spanning ratio of these graphs is at most  $1/(1 - 2 \sin(\theta/2))$ , when  $\theta < \pi/3$ , i.e. there are at least 7 cones.

In this paper, we look at the ordered variant of  $\theta$ -graphs. The ordered  $\theta$ -graph is constructed by inserting the vertices one by one and connecting each vertex to the closest previously-inserted vertex in each cone (a more precise definition follows in the next section). These graphs were introduced by Bose *et al.* [6] in order to construct spanners with nice additional properties, such as logarithmic maximum degree and logarithmic diameter. The current upper bound on the spanning ratio of these graphs is  $1/(1 - 2 \sin(\theta/2))$ , when  $\theta < \pi/3$ , i.e. there are at least 7 cones.

Recently, Bonichon *et al.* [3] showed that the unordered  $\theta_6$ -graph has spanning ratio 2. This was done by dividing the cones into two sets, positive and negative cones, such that each positive cone is adjacent to two negative cones and vice versa. It was shown that when edges are added only in the positive cones, in which case the graph is called the half- $\theta_6$ -graph, the resulting graph is equivalent to the TD-Delaunay triangulation (the Delaunay triangulation where the empty region is an equilateral triangle) whose spanning ratio is 2, as shown by Chew [9]. An alternative, inductive proof of the spanning ratio of the half- $\theta_6$ -graph was presented by Bose *et al.* [5]. This inductive proof was generalized to show that the  $\theta_{(4k+2)}$ -graph has spanning ratio  $1 + 2 \sin(\theta/2)$ , where  $k$  is an integer and at least 1. This spanning ratio is tight, i.e. there is a matching lower bound. Recently, the upper bounds on the spanning ratio of the  $\theta_{(4k+4)}$ -graph was improved to  $1 + 2 \sin(\theta/2)/(\cos(\theta/2) - \sin(\theta/2))$  and those of the  $\theta_{(4k+3)}$ -graph and the  $\theta_{(4k+5)}$ -graph were improved to  $\cos(\theta/4)/(\cos(\theta/2) - \sin(3\theta/4))$  [8].

By applying techniques similar to the ones used to improve the spanning ratio of unordered  $\theta$ -graphs, we improve the spanning ratio of the ordered  $\theta_{(4k+4)}$ -graph to  $1 + 2 \sin(\theta/2)/(\cos(\theta/2) - \sin(\theta/2))$  and show that this spanning ratio is tight. Unfortunately, this inductive proof cannot be applied to ordered  $\theta$ -graphs with an odd number of cones, as the triangle we apply induction on can

**Table 1.** An overview of upper and lower bounds on the spanning ratio of ordered  $\theta$ -graphs

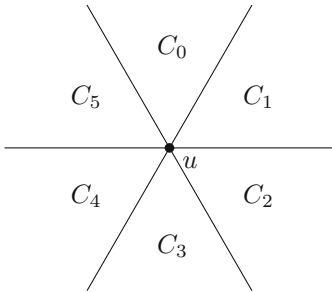
	Upper Bound	Lower Bound
$\theta_3, \theta_4, \theta_5$ , and $\theta_6$ -graph	-	Not spanners.
$\theta_{(4k+2)}$ -graph	$\frac{1}{1 - 2 \sin(\frac{\theta}{2})}$ , for $k \geq 2$ [6]	$\frac{1}{1 - 2 \sin(\frac{\theta}{2})}$
$\theta_{(4k+3)}$ -graph	$\frac{1}{1 - 2 \sin(\frac{\theta}{2})}$ [6]	$\frac{\cos(\frac{\theta}{4}) + \sin \theta}{\cos(\frac{3\theta}{4})}$
$\theta_{(4k+4)}$ -graph	$1 + \frac{2 \sin(\frac{\theta}{2})}{\cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})}$	$1 + \frac{2 \sin(\frac{\theta}{2})}{\cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})}$
$\theta_{(4k+5)}$ -graph	$\frac{1}{1 - 2 \sin(\frac{\theta}{2})}$ [6]	$1 + \frac{2 \sin(\frac{\theta}{2}) \cdot \cos(\frac{\theta}{4})}{\cos(\frac{\theta}{2}) - \sin(\frac{3\theta}{4})}$

become larger, depending on the order in which the vertices are inserted. We also show that the ordered  $\theta_{(4k+2)}$ -graph ( $k \geq 2$ ) has a tight spanning ratio of  $1/(1 - 2 \sin(\theta/2))$ .

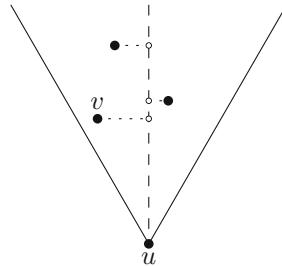
Next, we provide lower bounds for ordered  $\theta$ -graphs with  $4k + 3$  and  $4k + 5$  cones (see Table 1). For ordered  $\theta$ -graphs with  $4k + 2$  and  $4k + 5$  cones these lower bounds are strictly greater than the worst case spanning ratios of their unordered counterparts. Finally, we show that ordered  $\theta$ -graphs with 3, 4, 5, and 6 cones are not spanners. For the ordered  $\theta_3$ -graph this is not surprising, as its unordered counterpart is connected [1], but not a spanner [11]. For the ordered  $\theta_4$ ,  $\theta_5$ , and  $\theta_6$ -graph, however, this is a bit surprising since their unordered counterparts have recently been shown to be spanners [2, 3, 7]. In other words, we show, for the first time, that obtaining the nice additional properties of the ordered  $\theta$ -graphs comes at a price.

## 2 Preliminaries

We define a *cone*  $C$  to be a region in the plane between two rays originating from a vertex referred to as the apex of the cone. When constructing an (ordered)  $\theta_m$ -graph, for each vertex  $u$  consider the rays originating from  $u$  with the angle between consecutive rays being  $\theta = 2\pi/m$ . Each pair of consecutive rays defines a cone. The cones are oriented such that the bisector of some cone coincides with the vertical halfline through  $u$  that lies above  $u$ . Let this cone be  $C_0$  of  $u$  and number the cones in clockwise order around  $u$  (see Figure 1). The cones around the other vertices have the same orientation as the ones around  $u$ . We write  $C_i^u$  to indicate the  $i$ -th cone of a vertex  $u$ . For ease of exposition, we only consider point sets in general position: no two points lie on a line parallel to one of the rays that define the cones and no two vertices lie on a line perpendicular to the bisector of a cone.



**Fig. 1.** The cones having apex  $u$  in the (ordered)  $\theta_6$ -graph



**Fig. 2.** Three previously-inserted vertices are projected onto the bisector of a cone of  $u$ . Vertex  $v$  is the closest vertex.

Given some ordering of the vertices, the ordered  $\theta_m$ -graph is constructed as follows: we insert the vertices in the order given by the ordering. When a vertex  $u$  is inserted, for each cone  $C_i$  of  $u$ , we add an edge from  $u$  to the closest

previously-inserted vertex in that cone, where distance is measured along the bisector of the cone (see Figure 2). Note that our general position assumption implies that each vertex adds at most one edge per cone to the graph. As the ordered  $\theta$ -graph depends on the ordering of the vertices, different orderings can produce different  $\theta$ -graphs.

Given a vertex  $w$  in cone  $C_i$  of vertex  $u$ , we define the *canonical triangle*  $T_{uw}$  as the triangle defined by the borders of  $C_i$  and the line through  $w$  perpendicular to the bisector of  $C_i$ . We use  $m$  to denote the midpoint of the side of  $T_{uw}$  opposite  $u$  and  $\alpha$  to denote the smaller unsigned angle between  $uw$  and  $um$  (see Figure 3). Note that for any pair of vertices  $u$  and  $w$ , there exist two canonical triangles:  $T_{uw}$  and  $T_{wu}$ .

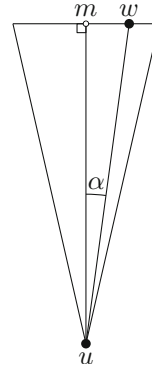


Fig. 3. The canonical triangle  $T_{uw}$

Before we bound the spanning ratios of ordered  $\theta$ -graphs, we first introduce a few useful geometric lemmas. Note that Lemmas 1 and 2 are proven in [8]. We use  $\angle xyz$  to denote the smaller angle between line segments  $xy$  and  $yz$ .

**Lemma 1.** *Let  $u, v$  and  $w$  be three vertices in the  $\theta_{(4k+x)}$ -graph, where  $x \in \{2, 3, 4, 5\}$ , such that  $w \in C_0^u$  and  $v \in T_{uw}$ , to the left of  $w$ . Let  $a$  be the intersection of the side of  $T_{uw}$  opposite to  $u$  with the left boundary of  $C_0^v$ . Let  $C_i^v$  denote the cone of  $v$  that contains  $w$  and let  $c$  and  $d$  be the upper and lower corner of  $T_{vw}$ . If  $1 \leq i \leq k - 1$ , or  $i = k$  and  $|cw| \leq |dw|$ , then  $\max\{|vc| + |cw|, |vd| + |dw|\} \leq |va| + |aw|$  and  $\max\{|cw|, |dw|\} \leq |aw|$ .*

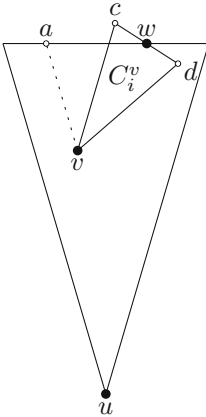


Fig. 4. The situation where we apply Lemma 1

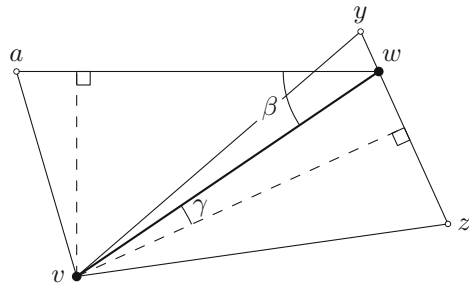


Fig. 5. The situation where we apply Lemma 2

**Lemma 2.** *Let  $u, v$  and  $w$  be three vertices in the  $\theta_{(4k+x)}$ -graph, such that  $w \in C_0^u, v \in T_{uw}$  to the left of  $w$ , and  $w \notin C_0^v$ . Let  $a$  be the intersection of the side of  $T_{uw}$  opposite to  $u$  with the left boundary of  $C_0^v$ . Let  $c$  and  $d$  be the corners of  $T_{vw}$  opposite to  $v$ . Let  $\beta = \angle awv$  and let  $\gamma$  be the unsigned angle between  $vw$  and the bisector of  $T_{vw}$ . Let  $c$  be a positive constant. If  $c \geq \frac{\cos \gamma - \sin \beta}{\cos(\frac{\theta}{2} - \beta) - \sin(\frac{\theta}{2} + \gamma)}$ , then  $\max\{|vc| + c \cdot |cw|, |vd| + c \cdot |dw|\} \leq |va| + c \cdot |aw|$ .*

### 3 The Ordered $\theta_{(4k+4)}$ -Graph

In this section, we give tight bounds on the spanning ratio of the ordered  $\theta_{(4k+4)}$ -graph, for any integer  $k \geq 1$ . We start by improving the upper bounds.

**Theorem 1.** *Let  $u$  and  $w$  be two vertices in the plane such that  $w$  was inserted before  $u$ . Let  $m$  be the midpoint of the side of  $T_{uw}$  opposite  $u$  and let  $\alpha$  be the unsigned angle between  $uw$  and  $um$ . There exists a path connecting  $u$  and  $w$  in the ordered  $\theta_{(4k+4)}$ -graph of length at most*

$$\left( \frac{\cos \alpha}{\cos(\frac{\theta}{2})} + c \cdot \left( \cos \alpha \cdot \tan\left(\frac{\theta}{2}\right) + \sin \alpha \right) \right) \cdot |uw|,$$

where  $c$  equals  $1/(\cos(\theta/2) - \sin(\theta/2))$ .

*Proof.* We assume without loss of generality that  $w \in C_0^u$ . We prove the theorem by induction on the rank, when ordered by area, of the canonical triangles  $T_{xy}$  for all pairs of vertices where  $y$  was inserted before  $x$ . Let  $a$  and  $b$  be the upper left and right corners of  $T_{uw}$ . Our inductive hypothesis is  $\delta(u, w) \leq \max\{|ua| + c \cdot |aw|, |ub| + c \cdot |bw|\}$ , where  $\delta(u, w)$  denotes the length of the shortest path from  $u$  to  $w$  in the ordered  $\theta_{(4k+4)}$ -graph and  $c$  equals  $1/(\cos(\theta/2) - \sin(\theta/2))$ .

We first show that this induction hypothesis implies the theorem. Basic trigonometry gives us the following equalities:  $|um| = |uw| \cdot \cos \alpha, |mw| = |uw| \cdot \sin \alpha, |am| = |bm| = |uw| \cdot \cos \alpha \cdot \tan(\theta/2)$ , and  $|ua| = |ub| = |uw| \cdot \cos \alpha / \cos(\theta/2)$ . Thus the induction hypothesis gives that  $\delta(u, w)$  is at most  $|uw| \cdot (\cos \alpha / \cos(\theta/2) + c \cdot (\cos \alpha \cdot \tan(\theta/2) + \sin \alpha))$ .

**Base Case:**  $T_{uw}$  has rank 1. Since this triangle is a smallest triangle where  $w$  was inserted before  $u$ , it is empty: if it is not empty, let  $x$  be a vertex in  $T_{uw}$ . Since  $T_{ux}$  and  $T_{xu}$  are both smaller than  $T_{uw}$ , the existence of  $x$  contradicts that  $T_{uw}$  is the smallest triangle where  $w$  was inserted before  $u$ . Since  $T_{uw}$  is empty,  $w$  is the closest vertex to  $u$  in  $C_0^u$ . Hence, since  $w$  was inserted before  $u$ ,  $u$  adds an edge to  $w$  when it is inserted. Therefore, the edge  $uw$  is part of the ordered  $\theta_{(4k+4)}$ -graph, and  $\delta(u, w) = |uw|$ . From the triangle inequality and the fact that  $c \geq 1$ , we have  $|uw| \leq \max\{|ua| + c \cdot |aw|, |ub| + c \cdot |bw|\}$ , so the induction hypothesis holds.

**Induction Step:** We assume that the induction hypothesis holds for all pairs of vertices with canonical triangles of rank up to  $j$ . Let  $T_{uw}$  be a canonical triangle of rank  $j + 1$ .

If  $uw$  is an edge in the ordered  $\theta_{(4k+4)}$ -graph, the induction hypothesis follows by the same argument as in the base case. If there is no edge between  $u$  and  $w$ , let  $v$  be the vertex in  $T_{uw}$  that  $u$  connected to when it was inserted, let  $a'$  and  $b'$  be the upper left and right corners of  $T_{uv}$ , and let  $a''$  be the intersection of the side of  $T_{uw}$  opposite  $u$  and the left boundary of  $C_0^v$  (see Figure 6).

We need to perform case distinction on whether  $w$  was inserted before or after  $v$ , to determine whether we can apply induction on  $T_{vw}$  or  $T_{vw}$ . Let  $c$  and  $d$  be the left and right corners of  $T_{vw}$  and let  $c'$  and  $d'$  be the left and right corner of  $T_{vw}$ . We note that since the ordered  $\theta_{(4k+4)}$ -graph has an even number of cones,  $vcwc'$  and  $vdwd'$  form two parallelograms. Thus, we have that  $|vc| + c \cdot |cw| = |wc'| + c \cdot |c'v|$  and  $|vd| + c \cdot |dw| = |wd'| + c \cdot |d'v|$ . Hence, we can assume without loss of generality that the canonical triangle we need to look at is  $T_{vw}$ .

Without loss of generality, we assume that  $v$  lies to the left of or has the same  $x$ -coordinate as  $w$ . Since we need to show that  $\delta(u, w) \leq \max\{|ua| + c \cdot |aw|, |ub| + c \cdot |bw|\}$ , it suffices to show that  $\delta(u, w) \leq |ua| + c \cdot |aw|$ . We perform a case analysis based on the cone of  $v$  that contains  $w$ : (a)  $w \in C_0^v$ , (b)  $w \in C_i^v$  where  $1 \leq i \leq k - 1$ , or  $i = k$  and  $|cw| \leq |dw|$ , (c)  $w \in C_k^v$  and  $|cw| > |dw|$ , (d)  $w \in C_{k+1}^v$ . To prove that  $\delta(u, w) \leq |ua| + c \cdot |aw|$ , it suffices to show that  $\delta(v, w) \leq |va''| + c \cdot |a''w|$ , as  $|uw| \leq |ua'| + c \cdot |a'v|$  and  $v, a'', a$ , and  $a'$  form a parallelogram (see Figure 6).

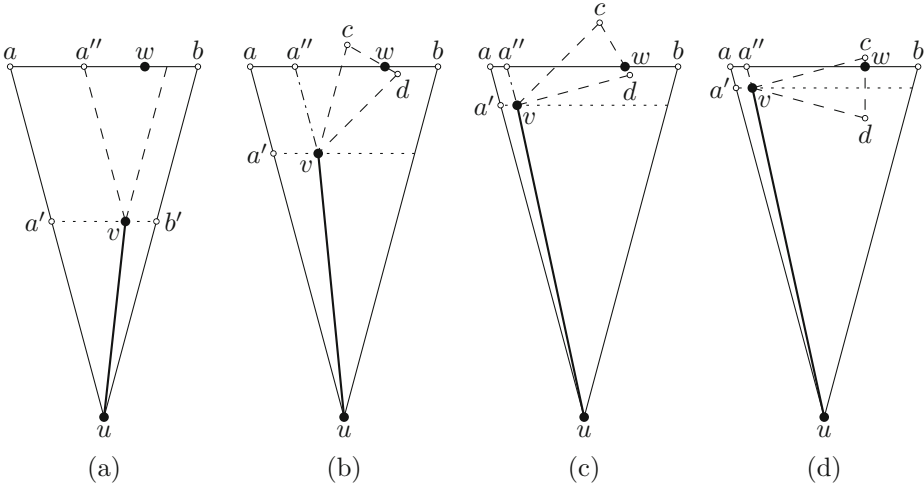


Fig. 6. The four cases based on the cone of  $v$  that contains  $w$

**Case (a):** Vertex  $w$  lies in  $C_0^v$  (see Figure 6a). Since  $T_{vw}$  has smaller area than  $T_{uw}$ , we apply the inductive hypothesis to  $T_{vw}$ . Since  $v$  lies to the left of or has the same  $x$ -coordinate as  $w$ , we have  $\delta(v, w) \leq |va''| + c \cdot |a''w|$ .

**Case (b):** Vertex  $w$  lies in  $C_i^v$ , where  $1 \leq i \leq k - 1$ , or  $i = k$  and  $|cw| \leq |dw|$ . Since  $T_{vw}$  is smaller than  $T_{uw}$ , by induction we have  $\delta(v, w) \leq \max\{|vc| + c \cdot |cw|, |vd| + c \cdot |dw|\}$  (see Figure 6b). Since  $w \in C_i^v$  where  $1 \leq$

$i \leq k - 1$ , or  $i = k$  and  $|cw| \leq |dw|$ , we can apply Lemma 1. Note that point  $a$  in Lemma 1 corresponds to point  $a''$  in this proof. Hence, we get that  $\max\{|vc| + |cw|, |vd| + |dw|\} \leq |va''| + |a''w|$  and  $\max\{|cw|, |dw|\} \leq |a''w|$ . Since  $c \geq 1$ , this implies that  $\max\{|vc| + c \cdot |cw|, |vd| + c \cdot |dw|\} \leq |va''| + c \cdot |a''w|$ .

**Case (c):** Vertex  $w$  lies in  $C_k^v$  and  $|cw| > |dw|$ . Since  $T_{vw}$  is smaller than  $T_{uw}$  and  $|cw| > |dw|$ , the induction hypothesis for  $T_{vw}$  gives  $\delta(v, w) \leq |vc| + c \cdot |cw|$  (see Figure 6c). Let  $\beta$  be  $\angle a''vw$  and let  $\gamma$  be the angle between  $vw$  and the bisector of  $T_{vw}$ . We note that  $\gamma = \theta - \beta$ . Hence Lemma 2 gives that  $|vc| + c \cdot |cw| \leq |va''| + c \cdot |a''w|$  holds when  $c \geq (\cos(\theta - \beta) - \sin \beta) / (\cos(\theta/2 - \beta) - \sin(3\theta/2 - \beta))$ . As this function is decreasing in  $\beta$  for  $\theta/2 \leq \beta \leq \theta$ , it is maximized when  $\beta$  equals  $\theta/2$ . Hence  $c$  needs to be at least  $(\cos(\theta/2) - \sin(\theta/2)) / (1 - \sin \theta)$ , which can be rewritten to  $1 / (\cos(\theta/2) - \sin(\theta/2))$ .

**Case (d):** Vertex  $w$  lies in  $C_{k+1}^v$  (see Figure 6d). Since  $T_{vw}$  is smaller than  $T_{uw}$ , we can apply induction on it. Since  $w$  lies above the bisector of  $T_{vw}$ , the induction hypothesis for  $T_{vw}$  gives  $\delta(v, w) \leq |vd| + c \cdot |dw|$ . Let  $\beta$  be  $\angle a''vw$  and let  $\gamma$  be the angle between  $vw$  and the bisector of  $T_{vw}$ . We note that  $\gamma = \beta$ . Hence Lemma 2 gives that  $|vd| + c \cdot |dw| \leq |va''| + c \cdot |a''w|$  holds when  $c \geq (\cos \beta - \sin \beta) / (\cos(\theta/2 - \beta) - \sin(\theta/2 + \beta))$ , which is equal to  $1 / (\cos(\theta/2) - \sin(\theta/2))$ .  $\square$

Since  $\cos \alpha / \cos(\theta/2) + (\cos \alpha \cdot \tan(\theta/2) + \sin \alpha) / (\cos(\theta/2) - \sin(\theta/2))$  is increasing for  $\alpha \in [0, \theta/2]$ , for  $\theta \leq \pi/4$ , it is maximized when  $\alpha = \theta/2$ , and we obtain the following corollary:

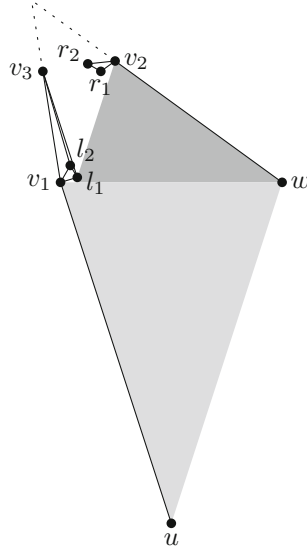
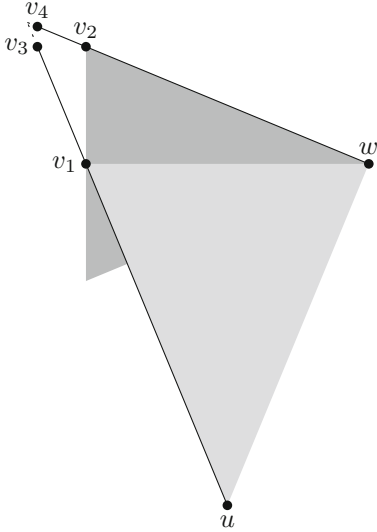
**Corollary 1.** *The ordered  $\theta_{(4k+4)}$ -graph ( $k \geq 1$ ) is a  $\left(1 + \frac{2 \sin(\frac{\theta}{2})}{\cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})}\right)$ -spanner.*

Next, we provide a matching lower bound.

**Lemma 3.** *The ordered  $\theta_{(4k+4)}$ -graph ( $k \geq 1$ ) has spanning ratio at least  $1 + \frac{2 \sin(\frac{\theta}{2})}{\cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})}$ .*

*Proof.* To prove the lower bound, we first construct a point set, after which we specify the order in which they are inserted into the graph. We place a vertex  $u$  and we place a vertex  $w$  arbitrarily close to the right boundary of  $C_0^u$ . Next, we place a vertex  $v_1$  arbitrarily close to the left corner of  $T_{uw}$ , followed by a vertex  $v_2$  arbitrarily close to the upper corner of  $T_{vv_1}$ . Finally, we repeat the following two steps an arbitrary number of times: we place a vertex  $v_i$  arbitrarily close to the left corner of  $T_{v_{i-2}v_{i-1}}$ , followed by a vertex  $v_{i+1}$  arbitrarily close to the upper corner of  $T_{v_{i-1}v_i}$ . Let  $v_n$  be the last vertex placed in this fashion. We insert the vertices in the following order:  $v_n, v_{n-1}, \dots, v_2, v_1, w, u$ . The resulting ordered  $\theta_{(4k+4)}$ -graph consists of a single path between  $u$  and  $w$  and is shown in Figure 7. Note that when a vertex  $v$  is inserted, all previously-inserted vertices lie in the same cone of  $v$ . This ensures that no shortcuts are introduced when inserting  $v$ .

We note that edges  $uv_1$  and edges of the form  $v_i v_{i+2}$  (for odd  $i \geq 1$ ) lie on a line. We also note that edges  $wv_2$  and edges of the form  $v_i v_{i+2}$  (for even  $i \geq 2$ ) lie on a line. Let  $x$  be the intersection of these two lines and let  $\beta$  be



**Fig. 7.** A lower bound for the ordered  $\theta_{(4k+4)}$ -graph

**Fig. 8.** A lower bound for the ordered  $\theta_{(4k+2)}$ -graph

$\angle xwv_1$ . Hence, as the number of vertices approaches infinity, the total length of the path approaches  $|ux| + |xw|$ . Using that  $\angle uxw = (\pi - \theta)/2 - \beta$ , we compute the following edge lengths:  $|ux| = |uw| \cdot \sin((\pi - \theta)/2 + \beta) / \sin((\pi - \theta)/2 - \beta)$  and  $|xw| = |uw| \cdot \sin \theta / \sin((\pi - \theta)/2 - \beta)$ . Since for the ordered  $\theta_{(4k+4)}$ -graph  $\beta = \theta/2$ , the sum of these equalities is  $1/(\cos \theta + \tan \theta)$ , which can be rewritten to  $1 + 2 \sin(\theta/2)/(\cos(\theta/2) - \sin(\theta/2))$ .  $\square$

**Theorem 2.** *The ordered  $\theta_{(4k+4)}$ -graph ( $k \geq 1$ ) has a tight spanning ratio of  $1 + \frac{2 \sin(\frac{\theta}{2})}{\cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})}$ .*

### 4 Lower Bounds

Next, we provide lower bounds for the ordered  $\theta_{(4k+2)}$ -graph, the ordered  $\theta_{(4k+3)}$ -graph, and the ordered  $\theta_{(4k+5)}$ -graph. For the ordered  $\theta_{(4k+2)}$ -graph, this lower bound implies that the current upper bound on the spanning ratio is tight. For the ordered  $\theta_{(4k+2)}$ -graph and the ordered  $\theta_{(4k+5)}$ -graph, these lower bounds are strictly larger than the upper bound on the worst case spanning ratio of its unordered counterpart.

**Lemma 4.** *The ordered  $\theta_{(4k+2)}$ -graph ( $k \geq 2$ ) has spanning ratio at least  $\frac{1}{1 - 2 \sin(\frac{\theta}{2})}$ .*

*Proof.* To prove the lower bound, we first construct a point set, after which we specify the order in which they are inserted into the graph. We place a vertex



$u$ , we place a vertex  $w$  arbitrarily close to the right boundary of  $C_0^u$ , and we place a vertex  $v_1$  arbitrarily close to the left corner of  $T_{uw}$ . Next, we place the following configuration an arbitrary number of times: place a vertex  $l_1$  in  $T_{v_1w}$  arbitrarily close to  $v_1$ , place a vertex  $v_2$  in the right corner of  $T_{wv_1}$ , place a vertex  $l_2$  close to the right boundary of  $T_{v_1v_2}$  arbitrarily close to  $v_1$ , place a vertex  $r_1$  in the intersection of  $T_{v_2l_2}$  and  $C_0^{l_2}$  arbitrarily close to  $v_2$ , place a vertex  $v_3$  in the left corner of the intersection of  $T_{l_1r_1}$  and  $T_{l_2r_1}$ , and place a vertex  $r_2$  in the intersection of  $T_{v_2v_3}$  and  $T_{v_3v_2}$  such that  $v_3r_2$  is parallel to  $v_1w$ . Since  $v_3r_2$  is parallel to  $v_1w$ , we can repeat placing this configuration, constructing a staircase of vertices (see Figure 8). When we place the  $i$ -th configuration, we place vertices  $l_{2i-1}$ ,  $v_{2i}$ ,  $l_{2i}$ ,  $r_{2i-1}$ ,  $v_{2i+1}$ , and  $r_{2i}$ . Let  $k$  be the total number of configurations.

We insert these vertices into the ordered  $\theta_{(4k+2)}$ -graph in the following order: starting from the  $k$ -th configuration down to the first one, insert the vertices of the  $i$ -th configuration in the order  $r_{2i}$ ,  $r_{2i-1}$ ,  $v_{2i+1}$ ,  $l_{2i}$ ,  $l_{2i-1}$ ,  $v_{2i}$ . Finally, we insert  $w$ ,  $v_1$ , and  $u$ . The resulting ordered  $\theta_{(4k+2)}$ -graph is essentially a path between  $u$  and  $w$  and is shown in Figure 8.

We note that edges  $uv_1$  and edges of the form  $v_iv_{i+2}$  (for odd  $i \geq 1$ ) lie on a line. We also note that edges  $wv_2$  and edges of the form  $v_iv_{i+2}$  (for even  $i \geq 2$ ) lie on a line. Let  $x$  be the intersection of these two lines. Hence, as the number of vertices approaches infinity, the total length of the path approaches  $|ux| + |xw|$ . Using that  $\angle xuw = \theta$ ,  $\angle xwu = (\pi + \theta)/2$ ,  $\angle xwv_2 = (\pi - 3\theta)/2$ , and the law of sines, we compute the following edge lengths:  $|ux| = |uw| \cdot \sin((\pi + \theta)/2) / \sin((\pi - 3\theta)/2)$  and  $|xw| = |uw| \cdot \sin \theta / \sin((\pi - 3\theta)/2)$ . Hence, the spanning ratio of the ordered  $\theta_{(4k+2)}$ -graph is at least  $(\sin((\pi + \theta)/2) + \sin \theta) / \sin((\pi - 3\theta)/2)$ , which can be rewritten to  $1/(1 - 2 \sin(\theta/2))$ .  $\square$

Since it is known that the  $\theta_{(4k+2)}$ -graph has a spanning ratio of at most  $1/(1 - 2 \sin(\theta/2))$  [6], this lower bound implies the following theorem.

**Theorem 3.** *The ordered  $\theta_{(4k+2)}$ -graph ( $k \geq 2$ ) has a tight spanning ratio of  $\frac{1}{1 - 2 \sin(\frac{\theta}{2})}$ .*

We also note that since the worst case spanning ratio of the unordered  $\theta_{(4k+2)}$ -graph is  $1 + 2 \sin(\theta/2)$  [4], this shows that the ordered  $\theta_{(4k+2)}$ -graph has a worse worst case spanning ratio.

**Lemma 5.** *The ordered  $\theta_{(4k+3)}$ -graph ( $k \geq 1$ ) has spanning ratio at least  $\frac{\cos(\frac{\theta}{4}) + \sin \theta}{\cos(\frac{3\theta}{4})}$ .*

*Proof.* The proof is analogous to the proof of Lemma 3, where  $\beta = \theta/4$ , and shows that the spanning ratio of the ordered  $\theta_{(4k+3)}$ -graph is at least  $(\sin(\pi/2 - \theta/4) + \sin \theta) / \sin(\pi/2 - 3\theta/4)$ , which can be rewritten to  $(\cos(\theta/4) + \sin \theta) / \cos(3\theta/4)$ .  $\square$

**Lemma 6.** *The ordered  $\theta_{(4k+5)}$ -graph ( $k \geq 1$ ) has spanning ratio at least  $1 + \frac{2 \sin(\frac{\theta}{2}) \cdot \cos(\frac{\theta}{4})}{\cos(\frac{\theta}{2}) - \sin(\frac{3\theta}{4})}$ .*

*Proof.* The proof is analogous to the proof of Lemma 3, where  $\beta = 3\theta/4$ , and shows that the spanning ratio of the ordered  $\theta_{(4k+5)}$ -graph is at least  $(\sin(\pi/2 + \theta/4) + \sin \theta) / \sin(\pi/2 - 5\theta/4)$ , which can be rewritten to  $1 + 2 \sin(\theta/2) \cdot \cos(\theta/4) / (\cos(\theta/2) - \sin(3\theta/4))$ .  $\square$

We note that this lower bound on the spanning ratio of the ordered  $\theta_{(4k+5)}$ -graph is the same as the current upper bound on  $\theta$ -routing on the unordered  $\theta_{(4k+5)}$ -graph, which is strictly greater than the current upper bound on the spanning ratio of the unordered  $\theta_{(4k+5)}$ -graph.

## 5 Ordered Theta-Graphs with Few Cones

In this section we show that ordered  $\theta$ -graphs with 3, 4, 5, or 6 cones are not spanners. For the ordered  $\theta_4$ ,  $\theta_5$ , and  $\theta_6$ -graph, this is surprising, since their unordered counterparts were recently show to be spanners [2,3,7].

For each of these ordered  $\theta$ -graphs, we build a tower similar to the ones from the previous section. However, unlike the towers in the previous section, the towers of ordered  $\theta$ -graphs that have at most 6 cones do not converge, thus giving rise to point sets where the spanning ratio depends on the size of these sets.

**Lemma 7.** *The ordered  $\theta_4$ -graph is not a spanner.*

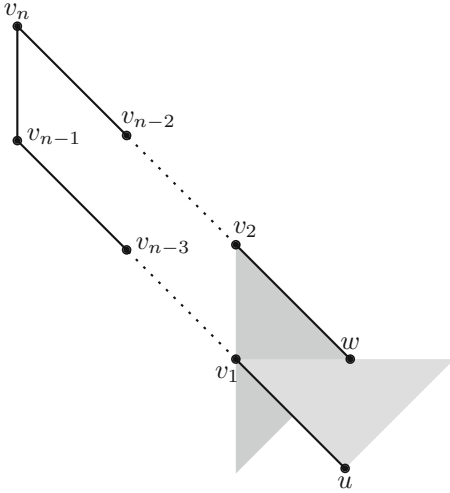
*Proof.* To prove that the ordered  $\theta_4$ -graph is not a spanner, we first construct a point set, after which we specify the order in which they are inserted into the graph. We place a vertex  $u$  and we place a vertex  $w$  slightly to the right of the bisector of  $C_0^u$ . Next, we place a vertex  $v_1$  arbitrarily close to the left corner of  $T_{uw}$  and we place a vertex  $v_2$  arbitrarily close to the upper corner of  $T_{wv_1}$ . Note that the placement of  $v_2$  implies that it lies slightly to the right of the bisector of  $C_0^{v_1}$ . Because of this, we can repeat placing pairs of vertices in a similar fashion, constructing a staircase of vertices (see Figure 9). Let  $v_n$  denote the last vertex that was placed.

We insert these vertices into the ordered  $\theta_4$ -graph in the following order:  $v_n, v_{n-1}, v_{n-2}, v_{n-3}, \dots, v_2, v_1, w, u$ . The resulting ordered  $\theta_4$ -graph consists of a single path between  $u$  and  $w$  and is shown in Figure 9.

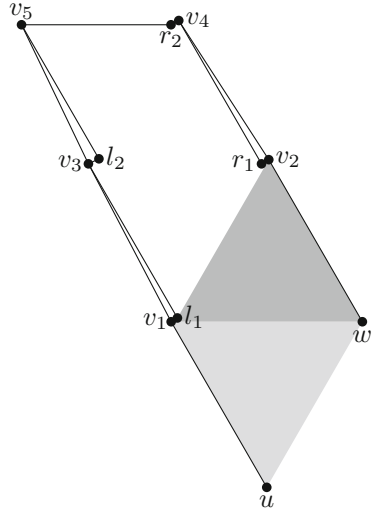
When we take  $|uw|$  to be 1, all diagonal edges have length  $c = \sqrt{2}$  and the total length of the path is  $1 + n \cdot \sqrt{2}$ . Hence, we have a graph whose spanning ratio depends on the number of vertices, implying that there does not exist a constant  $t$ , such that it is a  $t$ -spanner.  $\square$

**Lemma 8.** *The ordered  $\theta_3$ -graph is not a spanner.*

*Proof.* The proof is analogous to the proof of Lemma 7, where  $c = \cos(\pi/6) = \sqrt{3}/2$ , and shows that the total length of the path is  $1 + n \cdot \sqrt{3}/2$ . Hence, we have a graph whose spanning ratio depends on the number of vertices, implying that there does not exist a constant  $t$ , such that it is a  $t$ -spanner.  $\square$



**Fig. 9.** The ordered  $\theta_4$ -graph is not a spanner



**Fig. 10.** The ordered  $\theta_6$ -graph is not a spanner

**Lemma 9.** *The ordered  $\theta_5$ -graph is not a spanner.*

*Proof.* The proof is analogous to the proof of Lemma 7, where vertex  $w$  is placed such that the angle between  $uw$  and the bisector of  $C_0^u$  is  $\theta/4 = \pi/10$  and  $c = \cos(\pi/10)/\cos(\pi/5)$ , and shows that the total length of the path is  $1 + n \cdot \cos(\pi/10)/\cos(\pi/5)$ . Hence, we have a graph whose spanning ratio depends on the number of vertices, implying that there does not exist a constant  $t$ , such that it is a  $t$ -spanner. We note that the placement of  $v_i$  (for even  $i$ ) implies that the angle between  $v_{i-1}v_i$  and the bisector of  $C_0^{v_{i-1}}$  is  $\theta/4$ . Hence, every pair  $v_{i-1}, v_i$  of the staircase has the same relative configuration as the pair  $u, w$ .  $\square$

**Lemma 10.** *The ordered  $\theta_6$ -graph is not a spanner.*

*Proof.* To prove that the ordered  $\theta_6$ -graph is not a spanner, we first construct a point set, after which we specify the order in which they are inserted into the graph. We place a vertex  $u$ , we place a vertex  $w$  arbitrarily close to the right boundary of  $C_0^u$ , and we place a vertex  $v_1$  arbitrarily close to the left corner of  $T_{uw}$ . Next, we place the following configuration an arbitrary number of times: place a vertex  $l_1$  in  $T_{v_1w}$  arbitrarily close to  $v_1$ , place a vertex  $v_2$  in the right corner of  $T_{wl_1}$ , place a vertex  $r_1$  in  $T_{v_2l_1}$  arbitrarily close to  $v_2$ , and place a vertex  $v_3$  in the left corner of  $T_{l_1r_1}$ . Note that the line segment  $v_3r_1$  is parallel to  $v_1w$ . Because of this, we can repeat placing four vertices in a similar fashion, constructing a staircase of vertices (see Figure 10). When we place the  $i$ -th configuration, we place vertices  $l_i, v_{2i}, r_i$ , and  $v_{2i+1}$ . Let  $k$  be the total number of configurations we placed.

We insert these vertices into the ordered  $\theta_6$ -graph in the following order: starting from the  $k$ -th configuration down to the first one, insert the vertices of the  $i$ -th configuration in the order  $r_i, v_{2i+1}, l_i, v_{2i}$ . Finally, we insert  $w, v_1$ ,

and  $u$ . The resulting ordered  $\theta_6$ -graph is essentially a path between  $u$  and  $w$  and is shown in Figure 10.

When we take  $|uw|$  to be 1, we note that every configuration of four vertices extends the path length by 2. Hence, we have a graph whose spanning ratio depends on the number of vertices, implying that there does not exist a constant  $t$ , such that it is a  $t$ -spanner.  $\square$

## 6 Conclusion

We have provided tight spanning ratios for ordered  $\theta$ -graphs with  $4k + 2$  or  $4k + 4$  cones. We also provided lower bounds for ordered  $\theta$ -graphs with  $4k + 3$  or  $4k + 5$  cones. The lower bounds for ordered  $\theta$ -graphs with  $4k + 2$  or  $4k + 5$  cones are strictly greater than those of their unordered counterparts. Furthermore, we showed that ordered  $\theta$ -graphs with fewer than 7 cones are not spanners. For the ordered  $\theta_4$ ,  $\theta_5$ , and  $\theta_6$ -graph, this is surprising, since their unordered counterparts were shown to be spanners [2, 3, 7]. Thus we have shown for the first time that the nice properties obtained when using ordered  $\theta$ -graphs come at a price.

A number of open problems remain with respect to ordered  $\theta$ -graphs. For starters, though we provided lower bounds for ordered  $\theta$ -graphs with  $4k + 3$  or  $4k + 5$  cones, they do not match the current upper bound of  $1/(1 - 2\sin(\theta/2))$ . Hence, the obvious open problem is to find tight matching bounds for these graphs.

However, more importantly, there is currently no routing algorithm known for ordered  $\theta$ -graphs. The  $\theta$ -routing algorithm used for unordered  $\theta$ -graphs cannot be used, since it assumes that when there exist vertices in a cone of the current vertex, there also exists an edge to a vertex in that cone. This assumption does not need to hold for ordered  $\theta$ -graphs, since whether or not an edge is present depends on the order of insertion as well.

## References

1. Aichholzer, O., Bae, S.W., Barba, L., Bose, P., Korman, M., van Renssen, A., Taslakian, P., Verdonechot, S.: Theta-3 is connected. In: CCCG, pp. 205–210 (2013)
2. Barba, L., Bose, P., De Carufel, J.-L., van Renssen, A., Verdonechot, S.: On the stretch factor of the theta-4 graph. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 109–120. Springer, Heidelberg (2013)
3. Bonichon, N., Gavoille, C., Hanusse, N., Ilcinkas, D.: Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 266–278. Springer, Heidelberg (2010)
4. Bose, P., De Carufel, J.L., Morin, P., van Renssen, A., Verdonechot, S.: Optimal bounds on theta-graphs: More is not always better. In: CCCG, pp. 305–310 (2012)
5. Bose, P., Fagerberg, R., van Renssen, A., Verdonechot, S.: Competitive routing in the half- $\theta_6$ -graph. In: SODA, pp. 1319–1328 (2012)
6. Bose, P., Gudmundsson, J., Morin, P.: Ordered theta graphs. CGTA **28**(1), 11–18 (2004)

7. Bose, P., Morin, P., van Renssen, A., Verdonschot, S.: The  $\theta_5$ -graph is a spanner. In: Brandstädt, A., Jansen, K., Reischuk, R. (eds.) WG 2013. LNCS, vol. 8165, pp. 100–114. Springer, Heidelberg (2013)
8. Bose, P., van Renssen, A., Verdonschot, S.: On the spanning ratio of theta-graphs. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 182–194. Springer, Heidelberg (2013)
9. Chew, P.: There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences* **39**(2), 205–219 (1989)
10. Clarkson, K.: Approximation algorithms for shortest path motion planning. In: STOC, pp. 56–65 (1987)
11. El Molla, N.M.: Yao spanners for wireless ad hoc networks. Master’s thesis, Villanova University (2009)
12. Mark Keil, J.: Approximating the complete Euclidean graph. In: Karlsson, R., Lingas, A. (eds.) SWAT 1988. LNCS, vol. 318, pp. 208–213. Springer, Heidelberg (1988)
13. Narasimhan, G., Smid, M.: *Geometric Spanner Networks*. Cambridge University Press (2007)
14. Ruppert, J., Seidel, R.: Approximating the  $d$ -dimensional complete Euclidean graph. In: CCCG, pp. 207–210 (1991)

# Range Queries on Uncertain Data

Jian Li<sup>1</sup> and Haitao Wang<sup>2</sup>✉

<sup>1</sup> Institute for Interdisciplinary Information Sciences, Tsinghua University,  
Beijing 100084, China

lijian83@mail.tsinghua.edu.cn

<sup>2</sup> Department of Computer Science, Utah State University,  
Logan, UT 84322, USA

haitao.wang@usu.edu

**Abstract.** Given a set  $P$  of  $n$  uncertain points on the real line, each represented by its one-dimensional probability density function, we consider the problem of building data structures on  $P$  to answer range queries of the following three types: (1) top-1 query: find the point in  $P$  that lies in  $I$  with the highest probability, (2) top- $k$  query: given any integer  $k \leq n$  as part of the query, return the  $k$  points in  $P$  that lie in  $I$  with the highest probabilities, and (3) threshold query: given any threshold  $\tau$  as part of the query, return all points of  $P$  that lie in  $I$  with probabilities at least  $\tau$ . We present data structures for these range queries with linear or near linear space and efficient query time.

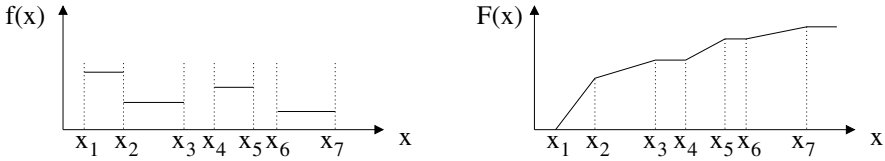
## 1 Introduction

In this paper, we study range queries on uncertain data. Let  $\mathbb{R}$  be any real line (e.g., the  $x$ -axis). In the (traditional) deterministic version of this problem, we are given a set  $P$  of  $n$  deterministic points on  $\mathbb{R}$ , and the goal is to build a data structure (also called “index” in database) such that given a range, specified by an interval  $I \subseteq \mathbb{R}$ , one point (or all points) in  $I$  can be retrieved efficiently. It is well known that a simple solution for this problem is a binary search tree over all points which is of linear size and can support logarithmic (plus output size) query time. However, in many applications, the location of each point may be uncertain and the uncertainty is represented in the form of probability distributions [3, 5, 11, 19, 20]. In particular, an *uncertain point*  $p$  is specified by its probability density function (pdf)  $f_p : \mathbb{R} \rightarrow \mathbb{R}^+ \cup \{0\}$ .

Let  $P$  be the set of  $n$  uncertain points in  $\mathbb{R}$  (with pdfs specified as input). Our goal is to build data structures to quickly answer range queries on  $P$ . In this paper, we consider the following three types of range queries, each of which involves a query interval  $I = [x_l, x_r]$ . For any point  $p \in P$ , we use  $\Pr[p \in I]$  to denote the probability that  $p$  is contained in  $I$ .

---

J. Li was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61202009, 61033001, 61361136003. H. Wang was supported in part by NSF under Grant CCF-1317143. Part of the work on this paper by H. Wang was carried out when he was visiting IIIS at Tsinghua University.



**Fig. 1.** The pdf of an uncertain point **Fig. 2.** The cdf of the uncertain point in Fig. 1

**Top-1 query:** Return the point  $p$  of  $P$  such that  $\Pr[p \in I]$  is the largest.

**Top- $k$  query:** Given any integer  $k$ ,  $1 \leq k \leq n$ , as part of the query, return the  $k$  points  $p$  of  $P$  such that  $\Pr[p \in I]$  are the largest.

**Threshold query:** Given a threshold  $\tau$ , as part of the query, return all points  $p$  of  $P$  such that  $\Pr[p \in I] \geq \tau$ .

We assume  $f_p$  is a step function, i.e., a *histogram* consisting of at most  $c$  pieces for some integer  $c \geq 1$  (e.g., see Fig. 1). More specifically,  $f_p(x) = y_i$  for  $x_{i-1} \leq x < x_i$ ,  $i = 1, \dots, c$ , with  $x_0 = -\infty$ ,  $x_c = \infty$ , and  $y_1 = y_c = 0$ . We assume  $c$  is a constant. The cumulative distribution function (cdf)  $F_p(x) = \int_{-\infty}^x f_p(t) dt$  is a monotone piecewise-linear function consisting of  $c$  pieces (e.g., see Fig. 2). Note that  $F_p(+\infty) = 1$ , and for any interval  $I = [x_l, x_r]$  the probability  $\Pr[p \in I]$  is  $F_p(x_r) - F_p(x_l)$ . As discussed in [2], the histogram model can be used to approximate most pdfs with arbitrary precision in practice, including the *discrete* pdf where each uncertain point can only appear in a finite number of locations.

We also study an important special case where the pdf  $f_p$  is a uniform distribution function, i.e.,  $f$  is associated with an interval  $[x_l(p), x_r(p)]$  such that  $f_p(x) = 1/(x_r(p) - x_l(p))$  if  $x \in [x_l(p), x_r(p)]$  and  $f_p(x) = 0$  otherwise. Clearly, the cdf  $F_p(x) = (x - x_l(p))/(x_r(p) - x_l(p))$  if  $x \in [x_l(p), x_r(p)]$ ,  $F_p(x) = 0$  if  $x \in (-\infty, x_l(p))$ , and  $F_p(x) = 1$  if  $x \in (x_r(p), +\infty)$ . Uniform distributions have been used as a major representation of uncertainty in some previous work (e.g., [10, 11, 16]). We refer to this special case the *uniform case* and the more general case where  $f_p$  is a histogram distribution function as the *histogram case*.

Throughout the paper, we will always use  $I = [x_l, x_r]$  to denote the query interval. The query interval  $I$  is *unbounded* if either  $x_l = -\infty$  or  $x_r = +\infty$ . For the threshold query, we will always use  $m$  to denote the output size of the query, i.e., the number of points  $p$  of  $P$  such that  $\Pr[p \in I] \geq \tau$ .

Range reporting on uncertain data has many applications [2, 11, 15, 18–20], As shown in [2], our problems are also useful even in some applications that involve only deterministic data. For example, consider the movie rating system in IMDB where each reviewer gives a rating from 1 to 10. A top- $k$  query on  $I = [7, +\infty)$  would find “the  $k$  movies such that the percentages of the ratings they receive at least 7 are the largest”; a threshold query on  $I = [7, +\infty)$  and  $\tau = 0.85$  would find “all the movies such that at least 85% of the ratings they receive are larger than or equal to 7”. Note that in the above examples the interval  $I$  is unbounded, and thus, it would also be interesting to have data structures particularly for quickly answering queries with unbounded query intervals.

## 1.1 Previous Work

The threshold query was first introduced by Cheng *et al.* [11]. Using R-trees, they [11] gave heuristic algorithms for the histogram case, without any theoretical performance guarantees. For the uniform case, if  $\tau$  is fixed for any query, they proposed a data structure of  $O(n\tau^{-1})$  size with  $O(\tau^{-1} \log n + m)$  query time [11]. These bounds depend on  $\tau^{-1}$ , which can be arbitrarily large.

Agarwal *et al.* [2] made a significant theoretical step on solving the threshold queries for the histogram case. If  $\tau$  is fixed, their approach can build an  $O(n)$  size data structure in  $O(n \log n)$  time, with  $O(m + \log n)$  query time. If the threshold  $\tau$  is not fixed, they built an  $O(n \log^2 n)$  size data structure in expected  $O(n \log^3 n)$  time that can answer each query in  $O(m + \log^3 n)$  time. Tao *et al.* [19, 20] considered the threshold queries in two and higher dimensions. They provided heuristic results and a query takes  $O(n)$  time in the worst case. Recently, Abdullah *et al.* [1] extended the notion of *geometric coresets* to uncertain data for range queries in order to obtain efficient approximate solutions.

As discussed in [2], our uncertain model is an analogue of the *attribute-level uncertainty model* in the probabilistic database literature. Another popular model is the *tuple-level uncertainty model* [5, 12, 21], where a tuple has fixed attribute values but its existence is uncertain. The range query under the latter model is much easier since a  $d$ -dimensional range searching over uncertain data can be transformed to a  $(d+1)$ -dimensional range searching problem over certain data [2, 21]. In contrast, the problem under the former model is more challenging, partly because it is unclear how to transform it to an instance on certain data.

## 1.2 Our Results

We say the *complexity* of a data structure is  $O(A, B)$  if it is of size  $O(B)$  and can be built in  $O(A)$  time. For the histogram case, we build data structures on  $P$  for answering queries with unbounded query intervals, and the complexities for the three type of queries are all  $O(n \log n, n)$ . The top-1 query time is  $O(\log n)$ ; the top- $k$  query time is  $O(k)$  if  $k = \Omega(\log n \log \log n)$  and  $O(\log n + k \log k)$  otherwise; the threshold query time is  $O(\log n + m)$ . Note that we consider  $c$  as a constant, otherwise all our results hold by replacing  $n$  by  $c \cdot n$ .

For the uniform case, we also present data structures for bounded query intervals. For the top-1 query, the complexity of our data structure is  $O(n \log n, n)$ , with query time  $O(\log n)$ . For other two queries, the data structure complexities are both  $O(n \log^2 n, n \log n)$ ; the top- $k$  query time is  $O(k)$  if  $k = \Omega(\log n \log \log n)$  and  $O(\log n + k \log k)$  otherwise, and the threshold query time is  $O(\log n + m)$ .

For the histogram case with bounded query intervals, Agarwal *et al.* [2] built a data structure of size  $O(n \log^2 n)$  in expected  $O(n \log^3 n)$  time, which can answer each threshold query in  $O(m + \log^3 n)$  time. Our results for the threshold queries are clearly better than the above solution for the uniform case and the histogram case with unbounded query intervals. Further, our algorithms are deterministic.

In Section 2, we give some observations. We discuss the uniform case and the histogram case in Sections 3 and 4, respectively. Due to the space limit, all lemma proofs are omitted and can be found in the full paper.



## 2 Preliminaries

For each uncertain point  $p$ , we call  $\Pr[p \in I]$  the  $I$ -probability of  $p$ . Let  $\mathcal{F}$  be the set of the cdfs of all points of  $P$ . Since each cdf is an increasing piecewise linear function, depending on the context,  $\mathcal{F}$  may also refer to the set of the  $O(n)$  line segments of all cdfs. Recall that  $I = [x_l, x_r]$  is the query interval.

**Lemma 1.** *If  $x_l = -\infty$ , then for any uncertain point  $p$ ,  $\Pr[p \in I] = F_p(x_r)$ .*

Let  $L$  be the vertical line with  $x$ -coordinate  $x_r$ . Since each cdf  $F_p$  is a monotonically increasing function, there is only one intersection between  $F_p$  and  $L$ . It is easy to know that for each cdf  $F_p$  of  $\mathcal{F}$ , the  $y$ -coordinate of the intersection of  $F_p$  and  $L$  is  $F_p(x_r)$ , which is the  $I$ -probability of  $p$  by Lemma 1. For each point in any cdf of  $\mathcal{F}$ , we call its  $y$ -coordinate the *height* of the point.

In the uniform case, each cdf  $F_p$  has three segments: the leftmost one is a horizontal segment with two endpoints  $(-\infty, 0)$  and  $(x_l(p), 0)$ , the middle one, whose slope is  $1/(x_r(p) - x_l(p))$ , has two endpoints  $(x_l(p), 0)$  and  $(x_r(p), 1)$ , and the rightmost one is a horizontal segment with two endpoints  $(x_r(p), 1)$  and  $(+\infty, 1)$ . We transform each  $F_p$  to the line  $l_p$  containing the middle segment of  $F_p$ . Consider an unbounded interval  $I$  with  $x_l = -\infty$ . We can use  $l_p$  to compute  $\Pr[p \in I]$  in the following way. Suppose the height of the intersection of  $L$  and  $l_p$  is  $y$ . Then,  $\Pr[p \in I] = 0$  if  $y < 0$ ,  $\Pr[p \in I] = y$  if  $0 \leq y \leq 1$ ,  $\Pr[p \in I] = 1$  if  $y > 1$ . Therefore, once we know  $l_p \cap L$ , we can obtain  $\Pr[p \in I]$  in constant time. Hence, we can use  $l_p$  instead of  $F_p$  to determine the  $I$ -probability of  $p$ . The advantage of using  $l_p$  is that lines are usually easier to deal with than line segments. Below, with a little abuse of notation, for the uniform case we simply use  $F_p$  to denote the line  $l_p$  for any  $p \in P$  and now  $\mathcal{F}$  is a set of lines.

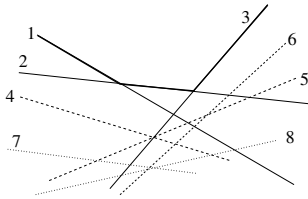
Fix the query interval  $I = [x_l, x_r]$ . For each  $i$ ,  $1 \leq i \leq n$ , denote by  $p_i$  the point of  $P$  whose  $I$ -probability is the  $i$ -th largest. Based on the above discussion, we obtain Lemma 2, which holds for both the histogram and uniform cases.

**Lemma 2.** *If  $x_l = -\infty$ , then for each  $1 \leq i \leq n$ ,  $p_i$  is the point of  $P$  such that  $L \cap F_{p_i}$  is the  $i$ -th highest among the intersections of  $L$  and all cdfs of  $\mathcal{F}$ .*

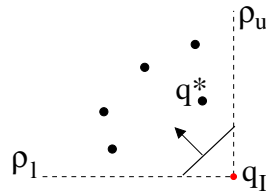
Suppose  $x_l = -\infty$ . Based on Lemma 2, to answer the top-1 query on  $I$ , it is sufficient to find the cdf of  $\mathcal{F}$  whose intersection with  $L$  is the highest; to answer the top- $k$  query, it is sufficient to find the  $k$  cdfs of  $\mathcal{F}$  whose intersections with  $L$  are the highest; to answer the threshold query on  $I$  and  $\tau$ , it is sufficient to find the cdfs of  $\mathcal{F}$  whose intersections with  $L$  have  $y$ -coordinates  $\geq \tau$ .

**Half-Plane Range Reporting.** As the half-plane range reporting data structure [9] is important for our later developments, we briefly discuss it in the dual setting. Let  $S$  be a set of  $n$  lines. Given any point  $q$ , the goal is to report all lines of  $S$  that are above  $q$ . An  $O(n)$ -size data structure can be built in  $O(n \log n)$  time that can answer each query in  $O(\log n + m')$  time, where  $m'$  is the number of lines above the query point  $q$  [9]. The data structure can be built as follows.

Let  $U_S$  be the upper envelope of  $S$  (e.g., see Fig. 3). We represent  $U_S$  as an array of lines  $l_1, l_2, \dots, l_h$  ordered as they appear on  $U_S$  from left to right. For



**Fig. 3.** Partitioning  $S$  into three layers:  $L_1(S) = \{1, 2, 3\}$ ,  $L_2(S) = \{4, 5, 6\}$ ,  $L_3(S) = \{7, 8\}$ . The thick polygonal chain is the upper envelope of  $S$ .



**Fig. 4.** Dragging a segment of slope 1 out of the corner at  $q_I$ :  $q^*$  is the first point that will be hit by the segment

each line  $l_i$ ,  $l_{i-1}$  is its *left neighbor* and  $l_{i+1}$  is its *right neighbor*. We partition  $S$  into a sequence  $L_1(S), L_2(S), \dots$ , of subsets, called *layers* (e.g., see Fig. 3). The first layer  $L_1(S) \subseteq S$  consists of the lines that appear on  $U_S$ . For  $i > 1$ ,  $L_i(S)$  consists of the lines that appear on the upper envelope of the lines in  $S \setminus \bigcup_{j=1}^{i-1} L_j(S)$ . Each layer  $L_i(S)$  is represented in the same way as  $U_S$ . To answer a half-plane range reporting query on a point  $q$ , let  $l(q)$  be the vertical line through  $q$ . We first determine the line  $l_i$  of  $L_1(S)$  whose intersection with  $l(q)$  is on the upper envelope of  $L_1(S)$ , by doing binary search on the array of lines of  $L_1(S)$ . Then, starting from  $l_i$ , we walk on the upper envelope of  $L_1(S)$  in both directions to report the lines of  $L_1(S)$  above the point  $q$ , in linear time with respect to the output size. Next, we find the line of  $L_2(S)$  whose intersection with  $l(q)$  is on the upper envelope of  $L_2(S)$ . We use the same procedure as for  $L_1(S)$  to report the lines of  $L_2(S)$  above  $q$ . Similarly, we continue on the layers  $L_3(S), L_4(S), \dots$ , until no line is reported in a certain layer. By using fractional cascading [7], after determining the line  $l_i$  of  $L_1(S)$  in  $O(\log n)$  time by binary search, the data structure [9] can report all lines above  $q$  in constant time each.

For any vertical line  $l$ , for each layer  $L_i(S)$ , denote by  $l_i(l)$  the line of  $L_i(S)$  whose intersection with  $l$  is on the upper envelope of  $L_i(S)$ . By fractional cascading [7], we have the following lemma for the data structure [9].

**Lemma 3.** [7,9] *For any vertical line  $l$ , after the line  $l_1(l)$  is known, we can obtain the lines  $l_2(l), l_3(l), \dots$  in this order in  $O(1)$  time each.*

### 3 The Uniform Distribution

Recall that  $\mathcal{F}$  is a set of lines in the uniform distribution.

#### 3.1 Queries with Unbounded Intervals

We first discuss the unbounded case where  $I = [x_l, x_r]$  is unbounded and some techniques introduced here will also be used later for the bounded case. Without loss of generality, we assume  $x_l = -\infty$ , and the other case where  $x_r = +\infty$  can be solved similarly. Recall that  $L$  is the vertical line with  $x$ -coordinate  $x_r$ .

For top-1 queries, by Lemma 2, we only need to maintain the upper envelope of  $\mathcal{F}$ , which can be computed in  $O(n \log n)$  time and  $O(n)$  space. For each query, it is sufficient to determine the intersection of  $L$  with the upper envelope of  $\mathcal{F}$ , which can be done in  $O(\log n)$  time. Next, we consider top- $k$  queries.

Given  $I$  and  $k$ , by Lemma 2, it suffices to find the  $k$  lines of  $\mathcal{F}$  whose intersections with  $L$  are the highest, and we let  $\mathcal{F}_k$  denote the set of the above  $k$  lines. As preprocessing, we build the half-plane range reporting data structure (see Section 2) on  $\mathcal{F}$ , in  $O(n \log n)$  time and  $O(n)$  space. Suppose the layers of  $\mathcal{F}$  are  $L_1(\mathcal{F}), L_2(\mathcal{F}), \dots$ . In the sequel, we compute the set  $\mathcal{F}_k$ . Let the lines in  $\mathcal{F}_k$  be  $l^1, l^2, \dots, l^k$  ordered from top to bottom by their intersections with  $L$ .

Let  $l_i(L)$  be the line of  $L_i(\mathcal{F})$  which intersects  $L$  on the upper envelope of the layer  $L_i(\mathcal{F})$ , for  $i = 1, 2, \dots$ . We first compute  $l_1(L)$  in  $O(\log n)$  time by binary search on the upper envelope of  $L_1(\mathcal{F})$ . Clearly,  $l^1$  is  $l_1(L)$ . Next, we determine  $l^2$ . Let the set  $H$  consist of the following three lines:  $l_2(L)$ , the left neighbor (if any) of  $l_1(L)$  in  $L_1(\mathcal{F})$ , and the right neighbor (if any) of  $l_1(L)$  in  $L_1(\mathcal{F})$ .

**Lemma 4.**  *$l^2$  is the line in  $H$  whose intersection with  $L$  is the highest.*

We refer to  $H$  as the *candidate set*. By Lemma 4, we find  $l^2$  in  $H$  in  $O(1)$  time. We remove  $l^2$  from  $H$ , and below we insert at most three lines into  $H$  such that  $l^3$  must be in  $H$ . Specifically, if  $l^2$  is  $l_2(L)$ , we insert the following three lines into  $H$ :  $l_3(L)$ , the left neighbor of  $l_2(L)$ , and the right neighbor of  $l_2(L)$ . If  $l^2$  is the left (resp., right) neighbor  $l$  of  $l_1(L)$ , we insert the left (resp., right) neighbor of  $l$  in  $L_1(\mathcal{F})$  into  $H$ . By generalizing Lemma 4, we can show  $l^3$  must be in  $H$  (the details are omitted). We repeat the same algorithm until we find  $l^k$ . To facilitate the implementation, we use a heap to store the lines of  $H$  whose “keys” in the heap are the heights of the intersections of  $L$  and the lines of  $H$ .

**Lemma 5.** *The set  $\mathcal{F}_k$  can be found in  $O(\log n + k \log k)$  time.*

We can improve the algorithm to  $O(\log n + k)$  time by using the selection algorithm in [14] for sorted arrays. The key idea is that we can implicitly obtain  $2k$  sorted arrays of  $O(k)$  size each and  $\mathcal{F}_k$  can be computed by finding the largest  $k$  elements in these arrays. The result is given in Lemma 6 with details omitted.

**Lemma 6.** *The set  $\mathcal{F}_k$  can be found in  $O(\log n + k)$  time.*

*Remark:* The above builds a data structure of  $O(n \log n, n)$  complexity that can answer each top- $k$  query in  $O(\log n + k)$  time for the uniform unbounded case.

For the threshold query, we are given  $I$  and a threshold  $\tau$ . We again build the half-plane range reporting data structure on  $\mathcal{F}$ . To answer the query, as discussed in Section 2, we only need to find all lines of  $\mathcal{F}$  whose intersections with  $L$  have  $y$ -coordinates larger than or equal to  $\tau$ . We first determine the line  $l_1(L)$  by doing binary search on the upper envelope of  $L_1(\mathcal{F})$ . Then, by Lemma 3, we find all lines  $l_2(L), l_3(L), \dots, l_j(L)$  whose intersections have  $y$ -coordinates larger than or equal to  $\tau$ . For each  $i$  with  $1 \leq i \leq j$ , we walk on the upper envelope of  $L_i(\mathcal{F})$ , starting from  $l_i(L)$ , on both directions in time linear to the output size to find the lines whose intersections have  $y$ -coordinates larger than or equal to  $\tau$ . Hence, the running time for answering the query is  $O(\log n + m)$ .

### 3.2 Queries with Bounded Intervals

Now we assume  $I = [x_l, x_r]$  is bounded. Consider any point  $p \in P$ . Recall that  $p$  is associated with an interval  $[x_l(p), x_r(p)]$  in the uniform case. Depending on the positions of  $I = [x_l, x_r]$  and  $[x_l(p), x_r(p)]$ , we classify  $[x_l(p), x_r(p)]$  and the point  $p$  into the following three types with respect to  $I$ .

**L-type:**  $[x_l(p), x_r(p)]$  and  $p$  are L-type if  $x_l \leq x_l(p)$ .

**R-type:**  $[x_l(p), x_r(p)]$  and  $p$  are R-type if  $x_r \geq x_r(p)$ .

**M-type:**  $[x_l(p), x_r(p)]$  and  $p$  are M-type if  $I \subset (x_l(p), x_r(p))$ .

Denote by  $P_L$ ,  $P_R$ , and  $P_M$  the sets of all L-type, R-type, and M-type of points of  $P$ , respectively. In the following, for each kind of query, we will build an data structure such that the different types of points will be searched separately (note that we will not explicitly compute the three subsets  $P_L$ ,  $P_R$ , and  $P_M$ ). For each point  $p \in P$ , we refer to  $x_l(p)$  as the *left endpoint* of the interval  $[x_l(p), x_r(p)]$  and refer to  $x_r(p)$  as the *right endpoint*. For simplicity of discussion, we assume no two interval endpoints of the points of  $P$  have the same value.

**The Top-1 Queries.** For any point  $p \in P$ , denote by  $\mathcal{F}_r(p)$  the set of the cdfs of the points of  $P$  whose intervals have left endpoints larger than or equal to  $x_l(p)$ . Again, as discussed in Section 2 we transform each cdf of  $\mathcal{F}_r(p)$  to a line. We aim to maintain the upper envelope of  $\mathcal{F}_r(p)$  for each  $p \in P$ . If we compute the  $n$  upper envelopes explicitly, we would have an data structure of size  $\Omega(n^2)$ . To reduce the space, we choose to use the persistent data structure [13] to maintain them implicitly such that data structure size is  $O(n)$ . The details are given below.

We sort the points of  $P$  by the left endpoints of their intervals from left to right, and let the sorted list be  $p'_1, p'_2, \dots, p'_n$ . For each  $i$  with  $2 \leq i \leq n$ , observe that the set  $\mathcal{F}_r(p'_{i-1})$  has exactly one more line than  $\mathcal{F}_r(p'_i)$ . If we maintain the upper envelope of  $\mathcal{F}_r(p'_i)$  by a balanced binary search tree (e.g., a red-black tree), then by updating it we can obtain the upper envelope of  $\mathcal{F}_r(p'_{i-1})$  by an insertion and a number of deletions on the tree, and each tree operation takes  $O(\log n)$  time. An easy observation is that there are  $O(n)$  tree operations in total to compute the upper envelopes of all sets  $\mathcal{F}_r(p'_1), \mathcal{F}_r(p'_2), \dots, \mathcal{F}_r(p'_n)$ . Further, by making the red-black tree persistent [13], we can maintain all upper envelopes in  $O(n \log n)$  time and  $O(n)$  space. We use  $\mathcal{L}$  to denote the above data structure.

We can use  $\mathcal{L}$  to find the point of  $P_L$  with the largest  $I$ -probability in  $O(\log n)$  time, as follows. First, we find the point  $p'_i$  such that  $x_l(p'_{i-1}) < x_l \leq x_l(p'_i)$ . It is easy to see that  $\mathcal{F}_r(p'_i) = P_L$ . Consider the unbounded interval  $I' = (-\infty, x_r]$ . Consider any point  $p$  whose cdf is in  $\mathcal{F}_r(p'_i)$ . Due to  $x_l(p) \geq x_l$ , we can obtain that  $\Pr[p \in I] = \Pr[p \in I']$ . Hence, the point  $p$  of  $\mathcal{F}_r(p'_i)$  with the largest value  $\Pr[p \in I]$  also has the largest value  $\Pr[p \in I']$ . This implies that we can instead use the unbounded interval  $I'$  as the query interval on the upper envelope of  $\mathcal{F}_r(p'_i)$ , in the same way as in Section 3.1. The persistent data structure  $\mathcal{L}$  maintains the upper envelope of  $\mathcal{F}_r(p'_i)$  such that we can find in  $O(\log n)$  time the point  $p$  of  $\mathcal{F}_r(p'_i)$  with the largest value  $\Pr[p \in I']$ .

Similarly, we can build a data structure  $\mathcal{R}$  of  $O(n)$  space in  $O(n \log n)$  time that can find the point of  $P_R$  with the largest  $I$ -probability in  $O(\log n)$  time.

To find the point of  $P_M$  with the largest  $I$ -probability, the approach for  $P_L$  and  $P_R$  does not work because we cannot reduce the query to another query with an unbounded interval. Instead, we reduce the problem to a “segment dragging query” by dragging a line segment out of a corner in the plane, as follows.

For each point  $p$  of  $P$ , we define a point  $q = (x_l(p), x_r(p))$  in the plane, and we say that  $p$  corresponds to  $q$ . Similar transformation was also used in [11]. Let  $Q$  be the set of the  $n$  points defined by the points of  $P$ . For the query interval  $I = [x_l, x_r]$ , we also define a point  $q_I = (x_l, x_r)$  (this is different from [11], where  $I$  defines a point  $(x_r, x_l)$ ). If we partition the plane into four quadrants with respect to  $q_I$ , then we have the following lemma.

**Lemma 7.** *The points of  $P_M$  correspond to the points of  $Q$  that strictly lie in the second quadrant (i.e., the northwest quadrant) of  $q_I$ .*

Let  $\rho_u$  be the upwards ray originating from  $q_I$  and let  $\rho_l$  be the leftwards ray originating from  $q_I$ . Imagine that starting from the point  $q_I$  and towards northwest, we drag a segment of slope 1 with two endpoints on  $\rho_u$  and  $\rho_l$  respectively, and let  $q^*$  be the point of  $Q$  hit first by the segment (e.g., see Fig. 4).

**Lemma 8.** *The point of  $P$  that defines  $q^*$  is in  $P_M$  and has the largest  $I$ -probability among all points in  $P_M$ .*

Based on Lemma 8, to determine the point of  $P_M$  with the largest  $I$ -probability, we only need to solve the above query on  $Q$  by dragging a segment out of a corner. More specifically, we need to build a data structure on  $Q$  to answer the following *out-of-corner segment-dragging queries*: Given a point  $q$ , find the first point of  $Q$  hit by dragging a segment of slope 1 from  $q$  and towards the northwest direction with the two endpoints on the two rays  $\rho_u(q)$  and  $\rho_l(q)$ , respectively, where  $\rho_u(q)$  is the upwards ray originating from  $q$  and  $\rho_l(q)$  is the leftwards ray originating from  $q$ . By using Mitchell’s result in [17] (reducing the problem to a point location problem), we can build an  $O(n)$  size data structure on  $Q$  in  $O(n \log n)$  time that can answer each such query in  $O(\log n)$  time.

Hence, for the uniform case, we can build in  $O(n \log n)$  time an  $O(n)$  size data structure on  $P$  that can answer each top-1 query in  $O(\log n)$  time.

**The Top- $k$  Queries.** To answer a top- $k$  query, we will do the following. First, we find the top- $k$  points in  $P_L$  (i.e., the  $k$  points of  $P_L$  whose  $I$ -probabilities are the largest), the top- $k$  points in  $P_R$ , and the top- $k$  points in  $P_M$ . Then, we find the top- $k$  points of  $P$  from the above  $3k$  points. Below we build three data structures for computing the top- $k$  points in  $P_L$ ,  $P_R$ , and  $P_M$ , respectively.

We first build the data structure for  $P_L$ . Again, let  $p'_1, p'_2, \dots, p'_n$  be the list of the points of  $P$  sorted by the left endpoints of their intervals from left to right. We construct a complete binary search tree  $T_L$  whose leaves from left to right store the  $n$  intervals of the points  $p'_1, p'_2, \dots, p'_n$ . For each internal node  $v$ , let  $P_v$  denote

the set of points whose intervals are stored in the leaves of the subtree rooted at  $v$ . We build the half-plane range reporting data structure discussed in Section 2 on  $P_v$ , denoted by  $D_v$ . Since the size of  $D_v$  is  $|P_v|$ , the total size of the data structure  $T_L$  is  $O(n \log n)$ , and  $T_L$  can be built in  $O(n \log^2 n)$  time.

We use  $T_L$  to compute the top- $k$  points in  $P_L$  as follows. By the standard approach and using  $x_l$ , we find in  $O(\log n)$  time a set  $V$  of  $O(\log n)$  nodes of  $T_L$  such that  $P_L = \bigcup_{v \in V} P_v$  and no node of  $V$  is an ancestor of another node. Then, we can determine the top- $k$  points of  $P_L$  in similarly as in Section 3.1. However, since we now have  $O(\log n)$  data structures  $D_v$ , we need to maintain the candidate sets for all such  $D_v$ 's. Specifically, after we find the top-1 point in  $D_v$  for each  $v \in V$ , we use a heap  $H$  to maintain them where the "keys" are the  $I$ -probabilities of the points. Let  $p$  be the point of  $H$  with the largest key. Clearly,  $p$  is the top-1 point of  $P_L$ ; assume  $p$  is from  $D_v$  for some  $v \in V$ . We remove  $p$  from  $H$  and insert at most three new points from  $D_v$  into  $H$ , in a similar way as in Section 3.1. We repeat the same procedure until we find all top- $k$  points of  $P_L$ .

To analyze the running time, for each node  $v \in V$ , we can determine in  $O(\log n)$  time the line in the first layer of  $D_v$  whose intersection with  $L$  is on the upper envelope of the first layer, and subsequent operations on  $D_v$  each takes  $O(1)$  time due to fractional cascading. Hence, the total time for this step in the entire algorithm is  $O(\log^2 n)$ . However, we can do better by building a fractional cascading structure [7] on the first layers of  $D_v$  for all nodes  $v$  of the tree  $T_L$ . In this way, the above step only takes  $O(\log n)$  time in the entire algorithm, i.e., do binary search only at the root of  $T_L$ . In addition, building the heap  $H$  initially takes  $O(\log n)$  time. Note that the additional fractional cascading structure on  $T_L$  does not change the size and construction time of  $T_L$  asymptotically [7]. The entire query algorithm has  $O(k)$  operations on  $H$  in total and the size of  $H$  is  $O(\log n + k)$ . Hence, the total time for finding the top- $k$  points of  $P_L$  is  $O(\log n + k \log(k + \log n))$ , which is  $O(\log n + k \log k)$  by Lemma 9.

**Lemma 9.**  $\log n + k \log(k + \log n) = O(\log n + k \log k)$ .

If  $k = \Omega(\log n \log \log n)$ , we have a better result in Lemma 10. Note that comparing with Lemma 6, we need to use other techniques to obtain Lemma 10 since the problem here involves  $O(\log n)$  half-plane range reporting data structures  $D_v$  while Lemma 6 only needs to deal with one such data structure.

**Lemma 10.** *If  $k = \Omega(\log n \log \log n)$ , we can compute the top- $k$  points in  $P_L$  in  $O(k)$  time.*

To compute the top- $k$  points of  $P_R$ , we build a similar data structure  $T_R$ , in a symmetric way as  $T_L$ , and we omit the details.

Finally, to compute the top- $k$  points in  $P_M$ , we do the following transformation. For each point  $p \in P$ , we define a point  $q = (x_l(p), x_r(p), 1/(x_r(p) - x_l(p)))$  in the 3-D space with  $x$ -,  $y$ -, and  $z$ -axes. Let  $Q$  be the set of all points in the 3-D space thus defined. Let the query interval  $I$  define an unbounded query box (or 3D rectangle)  $B_I = (-\infty, x_l) \times (x_r, +\infty) \times (-\infty, +\infty)$ . Similar to Lemma 7 in Section 3.1, the points of  $P_M$  correspond exactly to the points of  $Q \cap B_I$ . Further,

the top- $k$  points of  $P_M$  correspond to the  $k$  points of  $Q \cap B_I$  whose  $z$ -coordinates are the largest. Denote by  $Q_I$  the  $k$  points of  $Q \cap B_I$  whose  $z$ -coordinates are the largest. Below we build a data structure on  $Q$  for computing the set  $Q_I$  for any query interval  $I$  and thus finding the top- $k$  points of  $P_M$ .

We build a complete binary search tree  $T_M$  whose leaves from left to right store all points of  $Q$  ordered by the increasing  $x$ -coordinate. For each internal node  $v$  of  $T_M$ , we build an auxiliary data structure  $D_v$  as follows. Let  $Q_v$  be the set of the points of  $Q$  stored in the leaves of the subtree of  $T_M$  rooted at  $v$ . Suppose all points of  $Q_v$  have  $x$ -coordinates less than  $x_l$ . Let  $Q'_v$  be the points of  $Q_v$  whose  $y$ -coordinates are larger than  $x_r$ . The purpose of the auxiliary data structure  $D_v$  is to report the points of  $Q'_v$  in the decreasing  $z$ -coordinate order in constant time each after the point of  $q_v$  is found, where  $q_v$  is the point of  $Q'_v$  with the largest  $z$ -coordinate. To achieve this goal, we use the data structure given by Chazelle [8] (the one for Subproblem P1 in Section 5), and the data structure is a *hive graph* [6], which can be viewed as the preliminary version of the fractional cascading techniques [7]. By using the result in [8], we can build such a data structure  $D_v$  of size  $O(|Q_v|)$  in  $O(|Q_v| \log |Q_v|)$  time that can first compute  $q_v$  in  $O(\log |Q_v|)$  time and then report other points of  $Q'_v$  in the decreasing  $z$ -coordinate order in constant time each. Since the size of  $D_v$  is  $|Q_v|$ , the size of the tree  $T_M$  is  $O(n \log n)$ , and  $T_M$  can be built in  $O(n \log^2 n)$  time.

Using  $T_M$ , we find the set  $Q_I$  as follows. We first determine the set  $V$  of  $O(\log n)$  nodes of  $T_M$  such that  $\bigcup_{v \in V} Q_v$  consists of all points of  $Q$  whose  $x$ -coordinates less than  $x_l$  and no point of  $V$  is an ancestor of another point of  $V$ . Then, for each node  $v \in V$ , by using  $D_v$ , we find  $q_v$ , i.e., the point of  $Q_v$  with the largest  $z$ -coordinate, and insert  $q_v$  into a heap  $H$ , where the key of each point is its  $z$ -coordinate. We find the point in  $H$  with the largest key and remove it from  $H$ ; denote the above point by  $q'_1$ . Clearly,  $q'_1$  is the point of  $Q_I$  with the largest  $z$ -coordinate. Suppose  $q'_1$  is in a node  $v \in V$ . We proceed on  $D_v$  to find the point of  $Q_v$  with the second largest  $z$ -coordinate and insert it into  $H$ . Now the point of  $H$  with the largest key is the point of  $Q_I$  with the second largest  $z$ -coordinate. We repeat the above procedure until we find all  $k$  points of  $Q_I$ .

To analyze the query time, finding the set  $V$  takes  $O(\log n)$  time. For each node  $v \in V$ , the search for  $q_v$  on  $D_v$  takes  $O(\log n)$  time plus the time linear to the number of points of  $D_v$  in  $Q_I$ . Hence, the total time for searching  $q_v$  for all vertices  $v \in V$  is  $O(\log^2 n)$  time. Similarly as before, we can remove a logarithmic factor by building a fractional cascading structure on the nodes of  $T_M$  for searching such points  $q_v$ 's, in exactly the same way as in [6]. With the help of the fractional cascading structure, all these  $q_v$ 's for  $v \in V$  can be found in  $O(\log n)$  time. Note that building the fractional cascading structure does not change the construction time and the size of  $T_M$  asymptotically [6]. In addition, building the heap  $H$  initially takes  $O(\log n)$  time. In the entire algorithm there are  $O(k)$  operations on  $H$  in total and the size of  $H$  is always bounded by  $O(k + \log n)$ . Therefore, the running time of the query algorithm is  $O(\log n + k \log(k + \log n))$ , which is  $O(\log n + k \log k)$  by Lemma 9.

Using similar techniques as in Lemma 10, we obtain the following result.

**Lemma 11.** *If  $k = \Omega(\log n \log \log n)$ , we can compute the top- $k$  points in  $P_M$  in  $O(k)$  time.*

In summary, for the uniform case, we can build in  $O(n \log^2 n)$  time an  $O(n \log n)$  size data structure on  $P$  that can answer each top- $k$  query in  $O(k)$  time if  $k = \Omega(\log n \log \log n)$  and  $O(k \log k + \log n)$  time otherwise.

For the threshold queries, we build the same data structure as for the top- $k$  queries, i.e., the three trees  $T_L$ ,  $T_M$ , and  $T_R$ . The query algorithmic scheme is also similar. We omit the details.

## 4 The Histogram Distribution

In this section, we present our data structures for the histogram case, where  $I = [x_l, x_r]$  is unbounded. Again, we assume w.l.o.g. that  $x_l = -\infty$ . Recall that  $L$  is the vertical line with  $x$ -coordinate  $x_r$ . In the histogram case, the cdf of each point  $p \in P$  has  $c$  pieces; recall that we assumed  $c$  is a constant, and thus  $\mathcal{F}$  is still a set of  $O(n)$  line segments. Note that Lemmas 1 and 2 are still applicable.

For the top-1 queries, as in Section 3.1 it is sufficient to maintain the upper envelope of  $\mathcal{F}$ . Although  $\mathcal{F}$  now is a set of line segments, its upper envelope is still of size  $O(n)$  and can be computed in  $O(n \log n)$  time [4]. Given the query interval  $I$ , we can compute in  $O(\log n)$  time the cdf of  $\mathcal{F}$  whose intersection with  $L$  is on the upper envelope of  $\mathcal{F}$ .

For the threshold query, as discussed in Section 2 we only need to find the cdfs of  $\mathcal{F}$  whose intersections with  $L$  have  $y$ -coordinates at least  $\tau$ . Let  $q_I$  be the point  $(x_r, \tau)$  on  $L$ . A line segment is *vertically above*  $q_I$  if the segment intersects  $L$  and the intersection is at least as high as  $q_I$ . Hence, to answer the threshold query on  $I$ , it is sufficient to find the segments of  $\mathcal{F}$  that are vertically above  $q_I$ . Agarwal *et al.* [2] gave the following result on the *segment-below-point queries*. For a set  $S$  of  $O(n)$  line segments in the plane, a data structure of  $O(n)$  size can be computed in  $O(n \log n)$  time that can report the segments of  $S$  vertically below a query point  $q$  in  $O(m' + \log n)$  time, where  $m'$  is the output size. In our problem, we need a data structure on  $\mathcal{F}$  to solve the *segments-above-point queries*, which can be solved by using the similar approach as [2]. Therefore, we can build in  $O(n \log n)$  time an  $O(n)$  data structure on  $P$  that can answer each threshold query with an unbounded query interval in  $O(m + \log n)$  time.

For the top- $k$  queries, we only need to find the  $k$  segments of  $\mathcal{F}$  whose intersections with  $L$  are the highest. To this end, we can slightly modify the data structure for the segment-below-point queries in [2]. The details are omitted.

## References

1. Abdullah, A., Daruki, S., Phillips, J.: Range counting coresets for uncertain data. In: The 29th Symposium on Computational Geometry, SoCG, pp. 223–232 (2013)
2. Agarwal, P., Cheng, S.W., Tao, Y., Yi, K.: Indexing uncertain data. In: Proc. of the 28th Symposium on Principles of Database Systems, PODS, pp. 137–146 (2009)



3. Agarwal, P., Efrat, A., Sankararaman, S., Zhang, W.: Nearest-neighbor searching under uncertainty. In: Proc. of the 31st Symposium on Principles of Database Systems, PODS, pp. 225–236 (2012)
4. Agarwal, P., Sharir, M.: Red-blue intersection detection algorithms, with applications to motion planning and collision detection. *SIAM Journal on Computing* **19**, 297–321 (1990)
5. Agrawal, P., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: a system for data, uncertainty, and lineage. In: Proc. of the 32nd International Conference on Very Large Data Bases, VLDB, pp. 1151–1154 (2006)
6. Chazelle, B.: Filtering search: A new approach to query-answering. *SIAM Journal on Computing* **15**(3), 703–724 (1986)
7. Chazelle, B., Guibas, L.: Fractional cascading: I. A data structuring technique. *Algorithmica* **1**(1), 133–162 (1986)
8. Chazelle, B., Guibas, L.: Fractional cascading: II. Applications. *Algorithmica* **1**(1), 163–191 (1986)
9. Chazelle, B., Guibas, L., Lee, D.: The power of geometric duality. *BIT* **25**, 76–90 (1985)
10. Cheng, R., Chen, J., Mokbel, M., Chow, C.: Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In: IEEE International Conference on Data Engineering, ICDE, pp. 973–982 (2008)
11. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.: Efficient indexing methods for probabilistic threshold queries over uncertain data. In: Proc. of the 30th International Conference on Very Large Data Bases, VLDB, pp. 876–887 (2004)
12. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 523–544 (2006)
13. Driscoll, J., Sarnak, N., Sleator, D., Tarjan, R.: Making data structures persistent. *Journal of Computer and System Sciences* **38**(1), 86–124 (1989)
14. Frederickson, G., Johnson, D.: The complexity of selection and ranking in  $X + Y$  and matrices with sorted columns. *Journal of Computer and System Sciences* **24**(2), 197–208 (1982)
15. Knight, A., Yu, Q., Rege, M.: Efficient range query processing on uncertain data. In: Proc. of IEEE International Conference on Information Reuse and Integration, pp. 263–268 (2011)
16. Li, J., Deshpande, A.: Ranking continuous probabilistic datasets. *Proceedings of the VLDB Endowment* **3**(1), 638–649
17. Mitchell, J.:  $L_1$  shortest paths among polygonal obstacles in the plane. *Algorithmica* **8**(1), 55–88 (1992)
18. Qi, Y., Jain, R., Singh, S., Prabhakar, S.: Threshold query optimization for uncertain data. In: ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 315–326 (2010)
19. Tao, Y., Cheng, R., Xiao, X., Ngai, W., Kao, B., Prabhakar, S.: Indexing multidimensional uncertain data with arbitrary probability density functions. In: Proc. of the 31st International Conference on Very Large Data Bases, VLDB, pp. 922–933 (2005)
20. Tao, Y., Xiao, X., Cheng, R.: Range search on multidimensional uncertain data. *ACM Transactions on Database Systems (TODS)* **32** (2007)
21. Yiu, M., Mamoulis, N., Dai, X., Tao, Y., Vaitis, M.: Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. *IEEE Transactions of Knowledge Data Engineering (TKDE)* **21**, 108–122 (2009)

# On the Most Likely Voronoi Diagram and Nearest Neighbor Searching

Subhash Suri and Kevin Verbeek<sup>(✉)</sup>

Department of Computer Science, University of California, Santa Barbara, USA  
{suri,kverbeek}@cs.ucsb.edu

**Abstract.** We consider the problem of nearest-neighbor searching among a set of stochastic sites, where a stochastic site is a tuple  $(s_i, \pi_i)$  consisting of a point  $s_i$  in a  $d$ -dimensional space and a probability  $\pi_i$  determining its existence. The problem is interesting and non-trivial even in 1-dimension, where the *Most Likely Voronoi Diagram* (LVD) is shown to have worst-case complexity  $\Omega(n^2)$ . We then show that under more natural and less adversarial conditions, the size of the 1-dimensional LVD is significantly smaller: (1)  $\Theta(kn)$  if the input has only  $k$  distinct probability values, (2)  $O(n \log n)$  on average, and (3)  $O(n\sqrt{n})$  under smoothed analysis. We also present an alternative approach to the most likely nearest neighbor (LNN) search using *Pareto sets*, which gives a linear-space data structure and sub-linear query time in 1D for average and smoothed analysis models, as well as worst-case with a bounded number of distinct probabilities. Using the Pareto-set approach, we can also reduce the multi-dimensional LNN search to a sequence of nearest neighbor and spherical range queries.

## 1 Introduction

There is a growing interest in algorithms and data structures that deal with data uncertainty, driven in part by the rapid growth of unstructured databases where many attributes are missing or difficult to quantify [5, 6, 10]. Furthermore, an increasing amount of analytics today happens on data generated by machine learning systems, which is inherently probabilistic unlike the data produced by traditional methods. In computational geometry, the data uncertainty has typically been thought of as imprecision in the *positions* of objects—this viewpoint is quite useful for data produced by noisy sensors (e.g. LiDAR or MRI scanners) or associated with mobile entities, and many classical geometric problems including nearest-neighbors, convex hull, range searching and geometric optimization have been investigated in recent years [2–4, 14, 16–18].

Our focus, in this paper, is on a different form of uncertainty: each object’s location is known precisely but its *presence*, or activation, is subject to uncertainty. For instance, a company planning to open stores may know all the residents’ locations but has only a probabilistic knowledge about their interest in

---

The authors gratefully acknowledge support from the National Science Foundation, under the grants CNS-1035917 and CCF-11611495, and DARPA.

its products. Similarly, many phenomena where *influence* is transmitted through *physical proximity* involve entities whose positions are known but their ability to influence others is best modeled probabilistically: opinions, diseases, political views, etc. With this underlying motivation, we investigate one of the most basic *proximity* search problems for stochastic input.

Let a *stochastic site* be a tuple  $(s_i, \pi_i)$ , where  $s_i$  is a point in  $d$ -dimensional Euclidean space and  $\pi_i$  is the probability of its existence (namely, activation). Let  $\mathcal{S} = \{(s_1, \pi_1), (s_2, \pi_2), \dots, (s_n, \pi_n)\}$  be a set of stochastic sites, where we assume that the points  $s_i$ 's are distinct, and that the individual probabilities  $\pi_i$  are independent. Whenever convenient, we will simply use  $s_i$  to refer to the site  $(s_i, \pi_i)$ . We want to preprocess  $\mathcal{S}$  for answering *most likely nearest neighbor* (LNN) queries: a site  $s_i$  is the LNN of a query point  $q$  if  $s_i$  is present *and* all other sites closer than  $s_i$  to  $q$  are not present. More formally, let  $\bar{\pi}_i = 1 - \pi_i$ , and let  $B(q, s_i)$  be the set of sites  $s_j$  for which  $\|q - s_j\| < \|q - s_i\|$ . Then the probability that  $s_i$  is the LNN of  $q$  is  $\pi_i \times \prod_{s_j \in B(q, s_i)} \bar{\pi}_j$ . For ease of reference, we call this probability the *likeliness* of  $s_i$  with respect to  $q$ , and denote it as

$$\ell(s_i, q) = \pi_i \times \prod_{s_j \in B(q, s_i)} \bar{\pi}_j \tag{1}$$

The LNN of a query point  $q$  is the site  $s$  for which  $\ell(s, q)$  is maximized.

An important concept related to nearest neighbors is the Voronoi Diagram: it partitions the space into regions with the same nearest neighbor. In our stochastic setting, we seek the *most likely Voronoi Diagram* (LVD) of  $S$ : a partition of the space into regions so that all query points in a region have the same LNN. In addition to serving the role of a convenient *data structure* for LNN of query points, the structure of LVD also provides a compact representation of each stochastic site's region of *likely influence*.

**Related Work.** The topic of uncertain data has received a great deal of attention in recent years in the research communities of databases, machine learning, AI, algorithms and computational geometry. Due to limited space, we mention just a small number of papers that are directly relevant to our work. A number of researchers have explored nearest-neighbors and Voronoi diagrams for uncertain data [2, 4, 14], however, these papers focus on the *locational* uncertainty, with the goal of finding a neighbor minimizing the *expected* distance. In [19], Kamousi-Chan-Suri consider the stochastic (existence uncertainty) model but they also focus on the expected distance. Unfortunately, nearest neighbors under the expected measure can give non-sensical answers—a very low probability neighbor gets a large weight simply by being near the query point. Instead, the most likely nearest neighbor gives a more intuitive answer.

Over the past decade, smoothed analysis has emerged as a useful approach for analyzing problems in which the complexity of typical cases deviates significantly from the worst-case. A classical example is the Simplex algorithm whose worst-case complexity is exponential and yet it runs remarkably well on most practical instances of linear programming. The smoothed analysis framework proposed [22] offers a more insightful analysis than simple average case. Smoothed analysis is

also quite appropriate for many geometric problems [7, 8, 11, 12], because data is often the result of physical measurements that are inherently noisy.

**Our Results.** We first show that the most likely Voronoi diagram (LVD) has worst-case complexity  $\Omega(n^2)$  even in 1D, which is easily seen to be tight. We then show that under more natural, and less pathological, conditions the LVD has significantly better behavior. Specifically, (1) if the input has only  $k$  distinct probability values, then the LVD has size  $\Theta(nk)$ ; (2) if the probability values are randomly chosen (*average-case analysis*), then the LVD has expected size  $O(n \log n)$ ; (3) if the probability values (or the site positions) are worst-case but can be perturbed by some small value (*smoothed analysis*), then the LVD has size  $O(n\sqrt{n})$ . Of course, the LVD immediately gives an  $O(\log n)$  time data structure for LNN queries. Next, we propose an alternative data structure for LNN queries using Pareto sets. In 1-dimension, this data structure has linear size and answers LNN queries in worst-case  $O(k \log n)$  time when the input has only  $k$  distinct probability values, and in  $O(\log^2 n)$  and  $O(\sqrt{n} \log n)$  time under the average case and smoothed analysis models, respectively. Finally, the Pareto-set approach can be generalized to higher dimensions by reducing the problem to a sequence of nearest neighbor and spherical range queries. We give a concrete example of this generalization to finding the LNN in two dimensions.

## 2 The LVD Can Have Quadratic Complexity in 1D

The most likely nearest neighbor problem has non-trivial complexity even in the simplest of all settings: points on a line. Indeed, the LNN even violates a basic property often used in data structure design: *decomposability*. With deterministic data, one can split the input into a number of subsets, compute the nearest neighbor in each subset, and then choose the closest of those neighbors. As the following simple example shows, this basic property does not hold for the LNN.

Let the input have 3 sites  $\{(-2, \frac{1}{4}), (1, \frac{1}{3}), (3, \frac{3}{5})\}$ , and consider the query point  $q = 0$  (see Figure 1). Suppose we decompose the input into two subsets, sites to the left, and sites to the right of the query point. Then, it is easy to check that  $s_1$  is the LNN on the left, and  $s_3$  is the LNN for the right subset. However, the overall LNN of  $q$  turns out to be  $s_2$ , as is easily verified by the likeliness probabilities:  $\ell(s_1, q) = \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{6}$ ,  $\ell(s_2, q) = \frac{1}{3}$ , and  $\ell(s_3, q) = \frac{2}{3} \cdot \frac{3}{4} \cdot \frac{3}{5} = \frac{3}{10}$ .

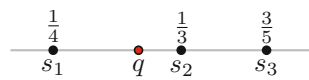
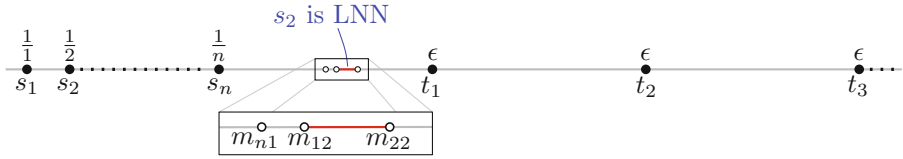


Fig. 1. The LNN of  $q$  is  $s_2$

The likeliness region for a site is also not necessarily connected: in fact, the following theorem shows that the LVD on a line can have quadratic complexity.

**Theorem 1.** *The most likely Voronoi diagram (LVD) of  $n$  stochastic sites on a line can have complexity  $\Omega(n^2)$ .*

*Proof.* Due to limited space, we sketch the main idea, deferring some of the technical details to the full version of the paper. The input for the lower bound consists of two groups of  $n$  sites each, for a total of  $2n$ . In the first group, called  $S$ , the  $i$ th



**Fig. 2.** The lower bound example of Theorem 1 with  $\Omega(n^2)$  complexity

site has position  $s_i = i/n$ , and probability  $\pi_i = 1/i$ , for  $i = 1, 2, \dots, n$ . In the second group, called  $T$ , the  $i$ th site has position  $t_i = i + 1$ , and probability  $\epsilon$ , for a choice of  $\epsilon$  specified later (see Figure 2). We will focus on the  $n^2$  midpoints  $m_{ij}$ , namely the bisectors, of pairs of sites  $s_i \in S$  and  $t_j \in T$ , and argue that the LNN changes in the neighborhood of each of these midpoints, proving the lower bound.

By construction, the midpoints  $m_{ij}$  are ordered lexicographically on the line, first by  $j$  and then by  $i$ . We will show that the LNN in the interval immediately to the left of the midpoint  $m_{ij}$  is  $s_i$ , which implies that the LVD has size  $\Omega(n^2)$ . In this proof sketch we assume that if two sites have the same likeliness then the site with the lower index is chosen as the LNN. Without this assumption the same bound can be obtained with a slightly altered construction, but the analysis becomes more complicated.

Let us consider a query point  $q$  that lies immediately to the left of the first midpoint  $m_{11}$ . It is easy to verify that  $\ell(s_i, q) = \frac{1}{n}$ , for all  $1 \leq i \leq n$ , and therefore  $s_1$  is  $q$ 's LNN. As the query point moves past  $m_{11}$ , only the likeliness of  $s_1$  changes to  $\frac{1-\epsilon}{n}$ , making  $s_2$  the LNN. The same argument holds as  $q$  moves past other midpoints towards the right, with the likeliness of corresponding sites changing to  $\frac{1-\epsilon}{n}$  in order, resulting in  $s_i$  becoming the new LNN when  $q$  lies just to the left of  $m_{i1}$ . After  $q$  passes  $m_{n1}$ , all sites of  $S$  have the same likeliness again, and the pattern is repeated for the remaining midpoints. To ensure that no site in  $T$  can ever be the LNN, we require that  $\frac{(1-\epsilon)^n}{n} > \epsilon$ , which holds for  $\epsilon = n^{-2}$ .  $\square$

### 3 Upper Bounds for the LVD in 1D

A matching upper bound of  $O(n^2)$  for the 1-dimensional LVD is easy: only the midpoints of pairs of sites can determine the boundary points of the LVD. In this section, we prove a number of stronger upper bounds, which may be more reflective of practical data sets. In particular, we show that if the number of distinct probability values among the stochastic sites is  $k$ , then the LVD has size  $\Theta(kn)$ , where clearly  $k \leq n$ . Thus, the LVD has size only  $O(n)$  if the input probabilities come from a fixed, constant size universe, not an unrealistic assumption in practice. Second, the lower bound construction of Theorem 1 requires a highly pathological arrangement of sites and their probabilities, unlikely to arise in practice. We therefore analyze the LVD complexity using average-case and smoothed analysis, and prove upper bounds of  $O(n \log n)$  and  $O(n\sqrt{n})$ , respectively.

#### 3.1 Structure of the LVD

We first establish some structural properties of the LVD; in particular, which midpoints (bisectors) form the boundaries between adjacent cells of the LVD.

For ease of reference, let us call these midpoints *critical*. Given a query point  $q$ , let  $\mathcal{L}(q)$  denote the sorted list of sites in  $\mathcal{S}$  by their (increasing) distance to  $q$ . Clearly, as long as the list  $\mathcal{L}(q)$  does not change by moving  $q$  along the line, its LNN remains unchanged. The order only changes at a midpoint  $m_{ij}$ , in which case  $s_i$  and  $s_j$  swap their positions in the list. The following lemmas provide a simple rule for determining critical midpoints.

**Lemma 1.** *Suppose that the midpoint  $m_{ij}$  of two sites  $s_i$  and  $s_j$  ( $s_i < s_j$ ) is critical, and consider the points  $q'$  immediately to the left of  $m_{ij}$ , and  $q''$  immediately to the right of  $m_{ij}$ . Then, either  $s_i$  is the LNN of  $q'$ , or  $s_j$  is the LNN of  $q''$ .*

*Proof.* Suppose, for the sake of contradiction, that the LNN of  $q'$  is not  $s_i$ , but instead some other site  $s_z$ . Consider the list  $\mathcal{L}(q')$  of sites ordered by their distance to the query, and consider the change to this list as the query point shifts from  $q'$  to  $q''$ . The only change is swapping of  $s_i$  and  $s_j$ . Then the likelihood of  $s_i$  and  $s_j$  satisfy  $\ell(s_i, q'') < \ell(s_i, q')$  and  $\ell(s_j, q'') > \ell(s_j, q')$ , while for all other sites  $s$ , we have  $\ell(s, q') = \ell(s, q'')$ . Therefore, the LNN of  $q''$  is either  $s_j$  or  $s_z$ . If  $s_z$  is the LNN of  $q''$ , then  $m_{ij}$  is not critical (a contradiction). So  $s_j$  must be the LNN of  $q''$  satisfying the condition of the lemma.  $\square$

**Lemma 2.** *If the midpoint  $m_{ij}$  of sites  $s_i$  and  $s_j$ , for  $s_i < s_j$ , is critical, then there cannot be a site  $s_z$  with  $s_z \in [s_i, s_j]$  and  $\pi_z \geq \max(\pi_i, \pi_j)$ .*

*Proof.* Suppose, for the sake of contradiction, that such a site  $s_z$  exists. By the position of  $s_z$ , we must have  $\|s_z - m_{ij}\| < \min\{\|s_i - m_{ij}\|, \|s_j - m_{ij}\|\}$ , and the same also holds for any query point  $q$  arbitrary close to  $m_{ij}$ . Because  $\pi_z \geq \max(\pi_i, \pi_j)$ , we have  $\ell(s_z, q) > \ell(s_i, q)$  and  $\ell(s_z, q) > \ell(s_j, q)$ , implying that  $s_z$  is more likely than both  $s_i$  and  $s_j$  to be the nearest neighbor of any  $q$  arbitrary close to  $m_{ij}$ . By Lemma 1, however, if  $m_{ij}$  is critical, then there exists a  $q$  close to  $m_{ij}$  for which the LNN is either  $s_i$  or  $s_j$ . Hence  $s_z$  cannot exist.  $\square$

### 3.2 Refined Upper Bounds

Our first result shows that if the stochastic input has only  $k$  distinct probabilities, then the LVD has size  $O(kn)$ . Let  $\{S_1, \dots, S_k\}$  be the partition of the input so that each group has sites of the same probability, ordered by increasing probability; that is, any site in  $S_j$  has higher probability than a site in  $S_i$ , for  $j > i$ . We write  $n_i = |S_i|$ , where  $\sum_{i=1}^k n_i = n$ .

**Lemma 3.** *The LVD of  $n$  stochastic sites on a line, with at most  $k$  distinct probabilities, has complexity  $\Theta(kn)$ .*

*Proof.* The lower bound on the size follows from an easy modification of the construction in Theorem 1: we use only  $k - 1$  points for the left side of the construction. We now analyze the upper bound. Suppose the midpoint  $m_{ij}$  defined by two sites  $s_i \in S_a$  and  $s_j \in S_b$  is critical, where  $1 \leq a < b \leq k$ , and without loss of generality, assume that  $s_i$  lies to the left of  $s_j$ . The sites in  $S_b$  have higher

probability than those in  $S_a$ , because of our assumption that  $a < b$ . Hence, by Lemma 2, there cannot be a site  $s \in S_b$  such that  $s \in [s_i, s_j]$ . By the same reasoning, the midpoint of  $s_i$  and a site  $s \in S_b$  with  $s > s_j$  also cannot be critical. Therefore,  $s_i$  can form critical midpoints with at most two sites in  $S_b$ : one on each side. Altogether,  $s_i$  can form critical midpoints with at most  $2k$  other sites  $s_j$  with  $\pi_j \geq \pi_i$ . Thus,  $|LVD| \leq 2k \sum_{i=1}^k n_i = 2kn$ .  $\square$

### 3.3 Average-Case and Smoothed Analysis of the LVD

We now show that even with  $n$  distinct probability values among the stochastic sites, the LVD has significantly smaller complexity as long as those probabilities are either assigned randomly to the points, or they can be perturbed slightly to get rid of the highly unstable pathological cases. More formally, for the average-case analysis we assume that we have a fixed set of  $n$  probabilities, and we randomly assign these probabilities to the sites. That is, we consider the average over all possible assignments of probabilities to sites. The smoothed analysis fixes a *noise* parameter  $a > 0$ , and draws a noise value  $\delta_i \in [-a, a]$  uniformly at random for each site  $(s_i, \pi_i)$ . This noise is used to perturb the input, either the location of a site or its probability. The location perturbation changes each site’s position to  $s'_i = s_i + \delta_i$ , resulting in the randomly perturbed input  $\mathcal{S}' = \{(s'_1, \pi_1), \dots, (s'_n, \pi_n)\}$ , which is a random variable. The smoothed complexity of the LVD is the expected complexity of the LVD of  $\mathcal{S}'$ , where we take the worst case over all inputs  $\mathcal{S}$ . The smoothed complexity naturally depends on the noise parameter  $a$ , which for the sake of simplicity we assume to be a constant—more detailed bounds involving  $a$  can easily be obtained. Of course, for this model we need to restrict the positions of sites to  $[0, 1]$ . The smoothed model perturbing the probabilities instead of the positions is defined analogously.

Our analysis uses a *partition tree*  $\mathcal{T}$  defined on the sites as follows. The tree is rooted at the site  $s_i$  with the highest probability. The remaining sites are split into a set  $S_1$ , containing the sites on the left of  $s_i$ , and a set  $S_2$  containing the rest (excluding  $s_i$ , see Figure 3 right). We then recursively construct the partition trees for  $S_1$  and  $S_2$ , whose roots become the children of  $s_i$ . (In case of ties, choose  $s_i$  to make the partition as balanced as possible.) The partition tree has the following useful property.

**Lemma 4.** *Let  $s_i$  and  $s_j$  be two sites with  $\pi_i \leq \pi_j$ . If the midpoint  $m_{ij}$  is critical, then  $s_j$  is an ancestor of  $s_i$  in  $\mathcal{T}$ .*

*Proof.* Let  $s_z$  be the lowest common ancestor of  $s_i$  and  $s_j$  in  $\mathcal{T}$ , assuming  $s_z \neq s_j$ . By construction,  $s_z \in [s_i, s_j]$  and  $\pi_z \geq \pi_j$ . Hence, by Lemma 2,  $m_{ij}$  cannot be critical.  $\square$

**Corollary 1.** *If the depth of  $\mathcal{T}$  is  $d$ , then the size of the LVD is  $O(dn)$ .*

Thus, we can bound the average and smoothed complexity of the LVD by analyzing the average and smoothed depth of the partition tree  $\mathcal{T}$ . In the average case,  $\mathcal{T}$  is essentially a *random binary search tree*. It is well known that

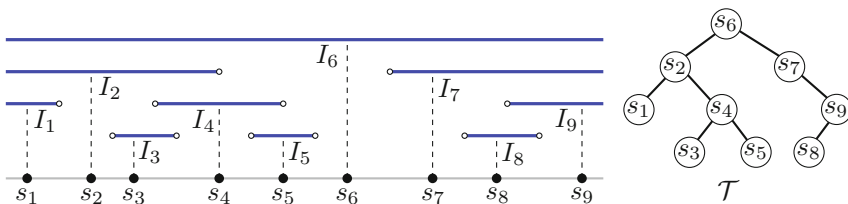
the depth of such a tree is  $O(\log n)$  (see e.g. [21]). In the smoothed model, if the perturbation is on the *position* of the sites, then a result by Manthey and Tantau [20, Lemma 10] shows that the smoothed depth of  $\mathcal{T}$  is  $O(\sqrt{n})$ .<sup>1</sup> We can easily extend that analysis to the perturbation on the probability values, instead of the positions of the sites. In a nutshell, the proof by Manthey and Tantau relies on the fact that the input elements can be partitioned into  $O(\sqrt{n}/\log n)$  groups such that the binary search tree of a single group is essentially random, and in this random tree, we can simply swap the roles of probabilities and positions. Thus, the smoothed depth of  $\mathcal{T}$  is also  $O(\sqrt{n})$  if the probabilities are perturbed. (If a perturbed probability falls outside  $[0, 1]$ , it is truncated, but the analysis holds due to our tie-breaking rule.)

**Theorem 2.** *Given a set of  $n$  stochastic sites on the line, its most likely Voronoi Diagram (LVD) has average-case complexity  $O(n \log n)$ , and smoothed complexity  $O(n\sqrt{n})$ .*

### 4 Algorithms for Constructing the LVD

Our main tool for constructing the LVD is the *likeliness curve*  $\ell(s_i): \mathbb{R} \rightarrow \mathbb{R}$  of a site  $s_i$ , which is simply the function  $\ell(s_i, q)$  with  $q$  ranging over the entire real line  $\mathbb{R}$ . A likeliness curve  $\ell(s_i)$  has  $O(n)$  complexity and it is a bimodal step function, achieving its maximum value at  $q = s_i$  (see Figure 4). By presorting all the sites in the left-to-right order, we can easily compute each  $\ell(s_i)$  in  $O(n)$  time, as follows. Start at  $q = s_i$  and walk to the left updating the value  $\ell(s_i, q)$  at every midpoint of the form  $m_{ij}$  with  $1 \leq j < i$ . We do the same for the right portion of  $\ell(s_i)$ , walking to the right instead (and  $i < j \leq n$ ). In the same way we can compute a restriction of  $\ell(s_i)$  to some interval  $I$ : assuming  $s_i \in I$ , it is easy to see that this restriction can be computed in time proportional to its complexity.

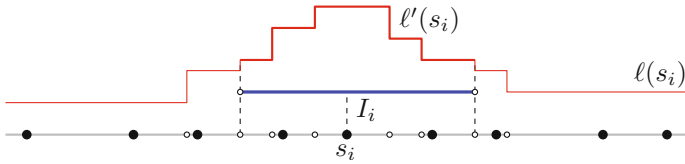
We can now compute the LVD by constructing the upper envelope  $\mathcal{U}$  of all  $\ell(s_i)$ , for  $i = 1, \dots, n$ . A naive construction, however, still takes  $O(n^2)$  time since the total complexity of all likeliness curves is quadratic. Instead, we restrict the likeliness curve of every site to a critical subpart such that the upper envelope



**Fig. 3.** The influence intervals (left) and the partition tree (right)

<sup>1</sup> In [20] a binary search tree is constructed from a sequence of real numbers. We obtain this sequence from our input by ordering the stochastic sites by decreasing probabilities. The construction of binary search trees in [20] then matches our construction of  $\mathcal{T}$ .





**Fig. 4.** The likelihood curve  $\ell(s_i)$  of  $s_i$  and its restriction  $\ell'(s_i)$  to  $I_i$

of these partial curves gives the correct  $\mathcal{U}$ . In particular, for each site  $s_i$ , define the *influence interval*  $I_i$  as follows. Let  $s_j$  be the first site encountered on the left of  $s_i$  for which  $\pi_j \geq \pi_i$ , and let  $s_z$  be such a site on the right side of  $s_i$ . Then we define  $I_i = [m_{ji}, m_{iz}]$ . (If  $s_j$  and/or  $s_z$  does not exist, we replace  $m_{ji}$  with  $-\infty$  and/or  $m_{iz}$  with  $\infty$ , respectively.) Observe that, for any  $q \notin I_i$ , either  $\ell(s_i, q) < \ell(s_j, q)$  or  $\ell(s_i, q) < \ell(s_z, q)$ , since either  $s_j$  or  $s_z$  is closer to  $q$  and  $\pi_j, \pi_z \geq \pi_i$ . We define  $\ell'(s_i)$  as the restriction of  $\ell(s_i)$  to the interval  $I_i$  (see Figure 4). Clearly,  $\mathcal{U}$  can be constructed by computing the upper envelope of just these restrictions  $\ell'(s_i)$ , and the complexity of each  $\ell'(s_i)$  is exactly the number of midpoints involving  $s_i$  that lie in  $I_i$ . Thus, given the defining sites  $s_j$  and  $s_z$  of  $I_i$ , the complexity of  $\ell'(s_i)$  is the number of sites in the interval  $[s_j, s_z]$  minus one (excluding  $s_i$ ).

**Lemma 5.** *The complexity of the union of all  $\ell'(s_i)$ , for  $i = 1, 2, \dots, n$ , is  $O(nd)$ , where  $d$  is the depth of the partition tree  $\mathcal{T}$  of the input sites. Furthermore, the union of  $\ell'(s_i)$  can be represented by  $d$  curves of  $O(n)$  complexity each.*

*Proof.* Let  $\sigma_1, \dots, \sigma_r$  be the set of sites at a fixed depth in the partition tree  $\mathcal{T}$  in order, and let  $\tau_i$ , for  $1 \leq i < r$ , be the lowest common ancestor of  $\sigma_i$  and  $\sigma_{i+1}$  in the tree. It is easy to see that the influence interval of a site  $\sigma_i$  is defined by a site in  $[\tau_{i-1}, \sigma_i]$  (possibly  $\tau_{i-1}$ ) and a site in  $[\sigma_i, \tau_i]$  (possibly  $\tau_i$ ), assuming  $1 < i < r$  (otherwise the influence interval may extend to  $-\infty$  or  $+\infty$ , see Figure 3). Hence the complexity of  $\ell'(\sigma_i)$  is bounded by the number of sites in the interval  $[\tau_{i-1}, \tau_i]$ . Furthermore, all influence intervals of the sites  $\sigma_1, \dots, \sigma_r$  are disjoint, and so we can combine all  $\ell'(\sigma_i)$  into a single curve with  $O(n)$  complexity. The result follows by constructing such a curve for each level of the partition tree. □

We can use Lemma 5 to efficiently compute the upper envelope  $\mathcal{U}$ . First, we compute the  $d$  curves  $f_1, \dots, f_d$  mentioned in Lemma 5, one for each level of  $\mathcal{T}$ . As we construct  $\mathcal{T}$ , we simultaneously compute  $\ell'(s_i)$  for each site  $s_i$ , in time  $O(|\ell'(s_i)|)$  time. This takes  $O(n)$  time per level of  $\mathcal{T}$ . We can then easily combine the individual parts  $\ell'(s_i)$  to obtain the curves  $f_1, \dots, f_d$ . The total running time of computing the curves  $f_1, \dots, f_d$  is  $O(n \log n + dn)$ .

Finally we can construct  $\mathcal{U}$  by computing the upper envelope of the curves  $f_1, \dots, f_d$ . We scan through the curves from left to right, maintaining two priority queues: (1) a priority queue for the events at which the curves change, and (2) a priority queue for maintaining the curve with the highest likelihood. Both priority queues have size  $d$ , which means that each event can be handled in  $O(\log d)$  time.

**Lemma 6.** *If  $d$  is the depth of  $\mathcal{T}$ , then the LVD can be constructed in  $O(n \log n + dn \log d)$  time.*

The algorithm is easily adapted for the case of  $k$  distinct probabilities. Consider the sites  $\sigma_1, \dots, \sigma_r$  (in order) for a single probability value. Since they all have the same probability, they bound each other’s influence intervals, and hence all influence intervals are interior disjoint. Now assume that a site  $s_j$  is contained in the interval  $[\sigma_i, \sigma_{i+1}]$ . Then  $s_j$  can add to the complexity of only  $\ell'(\sigma_i)$  and  $\ell'(\sigma_{i+1})$ , and no other  $\ell'(\sigma_z)$  with  $z \neq i, i + 1$ . Thus, we can combine the partial likeliness curves  $\ell'(\sigma_i)$  into a single curve of  $O(n)$  complexity. In total we obtain  $k$  curves of  $O(n)$  complexity each, from which we can construct the LVD.

**Theorem 3.** *The LVD of  $n$  stochastic sites in 1D can be computed in worst-case time  $O(n \log n + nk \log k)$  if the sites involve  $k$  distinct probabilities. Without the assumption on distinct probabilities, the construction takes  $O(n \log n \log \log n)$  time in the average case,<sup>2</sup> and  $O(n\sqrt{n} \log n)$  time in the smoothed analysis model.*

## 5 Time-Space Tradeoffs for LNN Searching

The worst-case complexity of the LVD is  $\Omega(n^2)$  even in 1 dimension and the Voronoi region of a single site can have  $\Omega(n)$  disjoint intervals. This raises a natural question: can the 1-dimensional LNN search be solved by a data structure of *subquadratic* size and *sub-linear* query time? While we cannot answer that question definitively, we offer an argument suggesting its hardness below.

### 5.1 A 3SUM Hard Problem

Consider the following problem, which we call the NEXT MIDPOINT PROBLEM: *given a set of  $n$  sites on a line, preprocess them so that for a query  $q$  we can efficiently compute the midpoint (for some pair of sites) that is immediately to the right of  $q$ .* The problem is inspired by the fact that an LNN query essentially needs to decide the location of the query point among the (potentially  $\Omega(n^2)$  critical) midpoints of the input. The following lemma proves 3SUM-hardness of this problem. (Recall that the 3SUM problem asks, given a set of numbers  $a_1, \dots, a_n$ , does there exist a triple  $(a_i, a_j, a_z)$  satisfying  $a_i + a_j + a_z = 0$ .)

**Lemma 7.** *Building the data structure plus answering  $2n$  queries of the NEXT MIDPOINT PROBLEM is 3SUM-hard.*

*Proof.* Consider an instance of the 3SUM problem consisting of numbers  $a_1, \dots, a_n$ . We use these numbers directly as sites for the NEXT MIDPOINT PROBLEM. If there exists a triple for which  $a_i + a_j + a_z = 0$ , then the midpoint  $m_{ij}$  is at  $-a_z/2$ . Thus, for every input number  $a_z$ , we query the NEXT MIDPOINT data structure just to the left and just to the right of  $-a_z/2$  (all numbers are integers, so this is easy). If the next midpoint is different for the two queries, then there exists a triple for which  $a_i + a_j + a_z = 0$ . Otherwise, such a triple does not exist.  $\square$

<sup>2</sup> In general,  $E[d \log d] \neq E[d] \log E[d]$ , but using the results of [13], we can easily show that  $E[d \log d] = O(\log n \log \log n)$  in our setting.

**Remark.** Thus, unless 3SUM can be solved in significantly faster than  $O(n^2)$  time, either the preprocessing time for the Next Midpoint problem is  $\Omega(n^2)$ , or that the query time is  $\Omega(n)$ . However, our reduction does not imply a hardness for the LNN problem in general: the order of the midpoints in the example of Theorem 1 follows a very simple pattern, which can be encoded efficiently.

### 5.2 LNN Search Using Pareto Sets

We now propose an alternative approach to LNN search using Pareto sets, which trades query time for space. Consider a query point  $q$ , and suppose that its LNN is the site  $s_i$ . Then,  $s_i$  must be Pareto optimal with respect to  $q$ , that is, there cannot be a site  $s_j$  closer to  $q$  with  $\pi_j \geq \pi_i$ . In fact, recalling the influence intervals  $I_i$  from the previous section, it is easy to check that  $s_i$  is Pareto optimal for  $q$  if and only if  $q \in I_i$ . This observation suggests the following algorithm for LNN: (1) compute the set  $S$  of sites  $s_i$  with  $q \in I_i$ , (2) compute  $\ell(s, q)$  for each  $s \in S$ , and (3) return  $s \in S$  with the maximum likeliness.

Step (1) requires computing the influence intervals for all sites, which is easily done as follows. Sort the sites in descending order of probability, and suppose they are numbered in this order. We incrementally add the sites to a balanced binary search tree, using the position of a site as its key. When we add a site  $s_i$  to the tree, all the sites with a higher probability are already in the tree. The interval  $I_i$  is defined by the two consecutive sites  $s_j$  and  $s_z$  in the tree such that  $s_i \in [s_j, s_z]$ . Thus, we can find  $s_j$  and  $s_z$  in  $O(\log n)$  time when adding  $s_i$  to the tree, and compute all the influence intervals in  $O(n \log n)$  total time.<sup>3</sup> To find the intervals containing the query point, we organize the influence intervals in an interval tree, which takes  $O(n \log n)$  time and  $O(n)$  space, and solves the query in  $O(\log n + r)$  time, where  $r$  is the output size. By the results in previous sections, we have  $r \leq \min\{k, d\}$ , where  $k$  is the number of distinct probabilities and  $d$  is the depth of  $\mathcal{T}$ .

Step (2) requires computing the likeliness of each site efficiently, and we do this by rewriting the likeliness function as follows:

$$\ell(s_i, q) = \pi_i \times \prod_{s_j \in (q-a, q+a)} \bar{\pi}_j \quad \text{where } a = |q - s_i| \quad (2)$$

With Equation (2), we can compute the likeliness of a site by a single range search query: an augmented balanced binary search tree, requiring  $O(n)$  space and  $O(n \log n)$  construction time, solves this query in  $O(\log n)$  time.

**Theorem 4.** *There is a data structure for 1D LNN search that needs  $O(n)$  space and  $O(n \log n)$  construction time and answers queries in (1) worst-case  $O(k \log n)$  time if the sites involve  $k$  distinct probabilities, (2) expected time  $O(\log^2 n)$  in the average case, and (3) expected time  $O(\sqrt{n} \log n)$  in the smoothed analysis model.*

<sup>3</sup> If there are sites with the same probability, we must first determine their influence intervals among sites with the same probability, before adding them to the tree. This can easily be achieved by first sorting the sites on position.

**Remark.** The query bounds of Theorem 4 for the average and smoothed analysis model are strong in the sense that they hold for *all* query points simultaneously, and not just for a fixed query point. That is, the bounds are for the expected worst case query time, rather than the expected query time.

## 6 The Pareto-Set Approach in Higher Dimensions

Our Pareto-set approach essentially requires the following two operations: (1) find the Pareto set for a query point  $q$ , and (2) compute the likeliness of a site w.r.t.  $q$ . In higher dimensions, the second operation can be performed with a spherical range query data structure, for which nearly optimal data structures exist [1]. The first operation can be reduced to a sequence of nearest neighbor queries, as follows: (1) find the nearest neighbor of  $q$ , say  $s_i$ , among all sites and add  $s_i$  to the Pareto set, (2) remove all sites with probability at most  $\pi_i$ , and (3) repeat steps (1) and (2) until no sites are left. We, therefore, need a data structure supporting the following query: *given a query point  $q$  and a probability  $\pi$ , find the closest site to  $q$  with probability higher than  $\pi$* . A dynamic nearest neighbor data structure can be adapted to answer this query as follows: incrementally add sites in decreasing order of probability, and make the data structure partially persistent. In this way, the data structure can answer the query we need, and partially persistent data structures often require only little extra space.

The required number of nearest neighbor and spherical range queries is precisely the number of elements in the Pareto set. For a query point  $q$ , consider the sequence of the sites' probabilities ordered by their increasing distance to  $q$ . Observe that the size of the Pareto set is precisely the number of left-to-right maxima in this sequence (see [20]). Therefore, the size of the Pareto set is (1) at most  $k$  when the input has at most  $k$  distinct probabilities, (2)  $O(\log n)$  in the average case model, and (3)  $O(\sqrt{n})$  in the smoothed analysis model. (Unlike the bound of Section 5.2, however, this result holds for any arbitrary query but not for all queries simultaneously.) A concrete realization of this abstract approach is discussed below for LNN search in 2D.

**2D Euclidean LNN Search.** For the sake of illustration, we consider only the average case of LNN queries. In this case, an incremental construction ordered by decreasing probabilities is simply a randomized incremental construction. We can then use the algorithm by Guibas *et al.* [15, Section 5] to incrementally construct the Voronoi diagram including a planar point location data structure, which uses  $O(n)$  space on average. Although not explicitly mentioned in [15], this data structure is partially persistent. Using this data structure we can answer a nearest neighbor query in  $O(\log^2 n)$  time. For the circular range queries, we use the data structure by Chazelle and Welzl [9, Theorem 6.1], which uses  $O(n \log n)$  space and can answer queries in  $O(\sqrt{n} \log^2 n)$  time. The final result is a data structure that uses, on average,  $O(n \log n)$  space and can answer LNN queries in  $O(\log^2 n \cdot \log n + \sqrt{n} \log^2 n \cdot \log n) = O(\sqrt{n} \log^3 n)$  time.

## 7 Concluding Remarks

The introduction of uncertainty seems to make even simple geometric problems quite hard, at least in the worst case. At the same time, uncertain data problems and algorithms may be particularly well-suited for average-case and smoothed analyses: after all, probabilities associated with uncertain data are inherently fuzzy measures, and problem instances whose answer changes dramatically with minor perturbations of input may suggest fragility of those probabilistic assumptions.

Our research suggests a number of open problems and research questions. In the 1-dimensional setting, we are able to settle the complexity of the LVD under all three analyses (average, smoothed, and worst-case), and it will be interesting to extend the results to higher dimensions. In particular, we believe the worst-case complexity of the  $d$ -dimensional LVD is  $\Omega(n^{2d})$ , but that is work in progress. Settling that complexity in the average or smoothed analysis case, as well as in the case of  $k$  distinct probabilities, is entirely open.

## References

1. Agarwal, P.: Range Searching. In: Goodman, J., O'Rourke, J. (eds.) *Handbook of Discrete and Computational Geometry*. CRC Press, New York (2004.)
2. Agarwal, P., Aronov, B., Har-Peled, S., Phillips, J., Yi, K., Zhang, W.: Nearest neighbor searching under uncertainty II. In: *Proc. 32nd PODS*, pp. 115–126 (2013)
3. Agarwal, P., Cheng, S., Yi, K.: Range searching on uncertain data. *ACM Trans. on Alg.* **8**(4), 43 (2012)
4. Agarwal, P., Efrat, A., Sankararaman, S., Zhang, W.: Nearest-neighbor searching under uncertainty. In: *Proc. 31st PODS*, pp. 225–236 (2012)
5. Aggarwal, C.: *Managing and Mining Uncertain Data*. *Advances in Database Systems*, 1st edn., vol. 35. Springer (2009)
6. Aggarwal, C., Yu, P.: A survey of uncertain data algorithms and applications. *IEEE Trans. Knowl. Data Eng.* **21**(5), 609–623 (2009)
7. de Berg, M., Haverkort, H., Tsirogiannis, C.: Visibility maps of realistic terrains have linear smoothed complexity. *J. of Comp. Geom.* **1**(1), 57–71 (2010)
8. Chaudhuri, S., Koltun, V.: Smoothed analysis of probabilistic roadmaps. *Comp. Geom. Theor. Appl.* **42**(8), 731–747 (2009)
9. Chazelle, B., Welzl, E.: Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.* **4**, 467–489 (1989)
10. Dalvi, N., Ré, C., Suciu, D.: Probabilistic databases: diamonds in the dirt. *Communications of the ACM* **52**(7), 86–94 (2009)
11. Damerow, V., Meyer auf der Heide, F., Räcke, H., Scheideler, C., Sohler, C.: Smoothed motion complexity. In: Di Battista, G., Zwick, U. (eds.) *ESA 2003*. LNCS, vol. 2832, pp. 161–171. Springer, Heidelberg (2003)
12. Damerow, V., Sohler, C.: Extreme points under random noise. In: Albers, S., Radzik, T. (eds.) *ESA 2004*. LNCS, vol. 3221, pp. 264–274. Springer, Heidelberg (2004)
13. Devroye, L.: A note on the height of binary search trees. *J. ACM* **33**(3), 489–498 (1986)

14. Evans, W., Sember, J.: Guaranteed Voronoi diagrams of uncertain sites. In: Proc. 20th CCCG, pp. 207–210 (2008)
15. Guibas, L., Knuth, D., Sharir, M.: Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica* **7**, 381–413 (1992)
16. Jørgensen, A., Löffler, M., Phillips, J.: Geometric computations on indecisive and uncertain points. CoRR, abs/1205.0273 (2012)
17. Löffler, M.: Data imprecision in computational geometry. PhD Thesis, Utrecht University (2009)
18. Löffler, M., van Kreveld, M.: Largest and smallest convex hulls for imprecise points. *Algorithmica* **56**, 235–269 (2010)
19. Kamousi, P., Chan, T., Suri, S.: Closest pair and the post office problem for stochastic points. *Comp. Geom. Theor. Appl.* **47**(2), 214–223 (2014)
20. Manthey, B., Tantau, T.: Smoothed analysis of binary search trees and quicksort under additive noise. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 467–478. Springer, Heidelberg (2008)
21. Reed, B.: The height of a random binary search tree. *J. ACM* **50**(3), 306–332 (2003)
22. Spielman, D., Teng, S.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM* **51**, 385–463 (2004)
23. Suri, S., Verbeek, K., Yıldız, H.: On the most likely convex hull of uncertain points. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 791–802. Springer, Heidelberg (2013)

# **Approximation Algorithms**

# An Improved Approximation Algorithm for the Minimum Common Integer Partition Problem

Weitian Tong and Guohui Lin<sup>(✉)</sup>

Department of Computing Science, University of Alberta,  
Edmonton, AB T6G 2E8, Canada  
{weitian,guohui}@ualberta.ca

**Abstract.** Given a collection of multisets  $\{X_1, X_2, \dots, X_k\}$  ( $k \geq 2$ ) of positive integers, a multiset  $S$  is a *common integer partition* for them if  $S$  is an integer partition of every multiset  $X_i, 1 \leq i \leq k$ . The *minimum common integer partition* ( $k$ -MCIP) problem is defined as to find a CIP for  $\{X_1, X_2, \dots, X_k\}$  with the minimum cardinality. We present a  $\frac{6}{5}$ -approximation algorithm for the 2-MCIP problem, improving the previous best algorithm of ratio  $\frac{5}{4}$  designed in 2006. We then extend it to obtain an absolute  $0.6k$ -approximation algorithm for  $k$ -MCIP when  $k$  is even (when  $k$  is odd, the approximation ratio is  $0.6k + 0.4$ ).

## 1 Introduction

The *minimum common integer partition* (MCIP) problem was introduced to the computational biology community by Chen *et al.* [7], formulated from their work on ortholog assignment and DNA fingerprint assembly. Mathematically, a *partition* of a positive integer  $x$  is a multiset  $\sigma(x) = \{a_1, a_2, \dots, a_t\}$  of positive integers such that  $a_1 + a_2 + \dots + a_t = x$ , where each  $a_i$  is called a *part* of the partition of  $x$  [2, 3]. For example,  $\{3, 2, 2, 1\}$  is a partition of  $x = 8$ ; so is  $\{6, 1, 1\}$ . A partition of a multiset  $X$  of positive integers is the multiset union of the partition  $\sigma(x)$  for all  $x$  of  $X$ , i.e.,  $\sigma(X) = \uplus_{x \in X} \sigma(x)$ . For example, as  $\{3, 2, 2, 1\}$  is a partition of  $x_1 = 8$  and  $\{3, 2\}$  is a partition of  $x_2 = 5$ ,  $\{3, 3, 2, 2, 2, 1\}$  is a partition for  $X = \{8, 5\}$ .

Given a collection of multisets  $\{X_1, X_2, \dots, X_k\}$  ( $k \geq 2$ ), a multiset  $S$  is a *common integer partition* (CIP) for them if  $S$  is an integer partition of every multiset  $X_i, 1 \leq i \leq k$ . For example, when  $k = 2$  and  $X_1 = \{8, 5\}$  and  $X_2 = \{6, 4, 3\}$ ,  $\{3, 3, 2, 2, 2, 1\}$  is a CIP for them since  $\{3, 3, 2, 2, 2, 1\}$  is also a partition for  $X_2 = \{6, 4, 3\}$ :  $3 + 3 = 6$ ,  $2 + 2 = 4$ , and  $2 + 1 = 3$ . The *minimum common integer partition* (MCIP) problem is defined as to find a CIP for  $\{X_1, X_2, \dots, X_k\}$  with the minimum cardinality, and it is denoted as  $\text{MCIP}(X_1, X_2, \dots, X_k)$ . For example, one can verify that, for the above  $X_1 = \{8, 5\}$  and  $X_2 = \{6, 4, 3\}$ ,  $\text{MCIP}(X_1, X_2) = \{6, 3, 2, 2\}$ . We use  $k$ -MCIP to denote the restricted version of the MCIP problem when the number of input multisets is fixed to be  $k$ .

For simplicity, we denote the *optimal*, i.e. the minimum cardinality, CIP for  $\{X_1, X_2, \dots, X_k\}$  as  $\text{OPT}(X_1, X_2, \dots, X_k)$ , or simply  $\text{OPT}$  when the input



multisets are clear from the context; analogously, we denote the CIP for  $\{X_1, X_2, \dots, X_k\}$  produced by an algorithm  $A$  as  $\text{CIP}_A(X_1, X_2, \dots, X_k)$ , or simply  $\text{CIP}_A$ ; without the algorithm subscript, we use CIP to denote any feasible common integer partition.

### 1.1 Known Results

For integer  $x \in \mathbb{Z}^+$ , its number of integer partitions increases very rapidly with  $x$ . For example, integer 3 has three partitions, namely  $\{3\}$ ,  $\{2, 1\}$ , and  $\{1, 1, 1\}$ ; integer 4 has five partitions, namely  $\{4\}$ ,  $\{3, 1\}$ ,  $\{2, 2\}$ ,  $\{2, 1, 1\}$ , and  $\{1, 1, 1, 1\}$ ; while integer 10 has 190,569,292 partitions according to [2].

Given a collection of multisets  $\{X_1, X_2, \dots, X_k\}$  ( $k \geq 2$ ), they have a CIP if and only if they have the same summation over their elements. Multisets with this property are called *related* [6], and we assume throughout the paper that the multisets in any instance of MCIP are related, as the verification takes only linear time.

One can see that the 2-MCIP problem generalizes the well-known subset sum problem [8], and thus it is NP-hard [6]. Chen *et al.* showed that 2-MCIP is APX-hard [6], via a linear reduction (also called an approximation preserving reduction) from the *maximum bounded 3-dimensional matching problem* [9].

Let  $m = |X_1| + |X_2| + \dots + |X_k|$  denote the total number of integers in the  $k$ -MCIP problem. For the positive algorithmic results, Chen *et al.* presented a linear time 2-approximation algorithm and an  $O(m^9)$ -time  $5/4$ -approximation algorithm for 2-MCIP [6], based on a heuristic for the *maximum weighted set packing problem* [9]. The  $5/4$ -approximation can be taken as a subroutine to design a  $0.625k$ -approximation algorithm for  $k$ -MCIP (when  $k$  is even; when  $k$  is odd, the approximation ratio is  $0.625k + 0.375$ ) [10]. Woodruff developed a framework for capturing the frequencies of the integers across the input multisets and presented a randomized  $O(m \log k)$ -time approximation algorithm for  $k$ -MCIP, with worst-case performance ratio  $0.6139k(1 + o(1))$  [10]. The basic idea is, when there are not too many distinct integers in the input multisets, most of the low frequency integers will have to be split into at least two parts in any common partition. Inspired by this idea, Zhao *et al.* [11] formulated the  $k$ -MCIP problem into a *flow decomposition* problem in an acyclic  $k$ -layer network with the goal to find a minimum number of directed simple paths from the source to the sink. Since this minimum number can be bounded by the number of arcs in the network according to the well-known *flow decomposition theorem* [1], Zhao *et al.* presented a scheme to reduce the number of arcs in the network, resulting in a de-randomized approximation algorithm with performance ratio  $0.5625k(1 + o(1))$ , which is the currently best.

### 1.2 Our Contributions

In this paper, we present a polynomial-time  $6/5$ -approximation algorithm for 2-MCIP. Subsequently, we obtain a  $0.6k$ -approximation algorithm for  $k$ -MCIP when  $k$  is even (when  $k$  is odd, the approximation ratio is  $0.6k + 0.4$ ). It is worth

pointing out that the ratio of  $0.5625k$  in [11] is asymptotic, that it holds for only sufficiently large  $k$ ; while our ratio of  $0.6k$  is absolute, that it holds for all  $k$ .

## 2 A 6/5-Approximation Algorithm for 2-MCIP

In this section, we deal with the 2-MCIP problem. For ease of presentation, we denote the two multisets of positive integers in an instance as  $X = \{x_1, x_2, \dots, x_m\}$  and  $Y = \{y_1, y_2, \dots, y_n\}$ , and assume without loss of generality that they are related. Recall that,  $\text{OPT}(X, Y)$  denotes the optimal solution — the minimum cardinality CIP for  $\{X, Y\}$ , and  $\text{CIP}_A(X, Y)$  denotes the solution CIP produced by algorithm  $A$ .

### 2.1 Preliminaries

Chen *et al.* presented a simple linear time 2-approximation algorithm for 2-MCIP [5,6], denoted as Apx21. In each iteration of Apx21, it chooses an (arbitrary) element  $x \in X$  and an (arbitrary) element  $y \in Y$ , and adds  $\min\{x, y\}$  to the solution  $\text{CIP}_{\text{Apx21}}$ ; subsequently, if  $x = y$  then  $x$  is removed from  $X$  and  $y$  is removed from  $Y$ ; otherwise  $\min\{x, y\}$  is removed from the multiset it appears in and  $\max\{x, y\}$  is replaced with  $\max\{x, y\} - \min\{x, y\}$  in the other multiset. Its performance ratio of 2 is seen from the fact that  $|\text{OPT}(X, Y)| \geq \max\{m, n\}$  and that the solution  $\text{CIP}_{\text{Apx21}}$  contains no more than  $m + n - 1$  integers. Consequently, we have the following lemma.

**Lemma 1.** [5,6]  $\max\{m, n\} \leq |\text{OPT}(X, Y)| \leq |\text{CIP}_{\text{Apx21}}| \leq m + n - 1$ .

Given an instance  $\{X, Y\}$  of 2-MCIP and an arbitrary CIP that specifies the integer partitions for all elements of  $X$  and  $Y$ , we say that  $x_i \in X$  is *mapped* to  $y_j \in Y$  if there exists an element of CIP that is a part of the partition for  $x_i$  and is also a part of the partition for  $y_j$ . This mapping relationship gives rise naturally to a bipartite graph  $G(X, Y)$ , in which the two disjoint subsets of vertices are  $X$  and  $Y$ , respectively, and vertex  $x_i$  and vertex  $y_j$  are adjacent if and only if  $x_i$  is mapped to  $y_j$  according to the CIP. Note that an edge of the bipartite graph  $G$  one-to-one corresponds to an element of CIP, and there could be multiple edges between a pair of vertices in  $G(X, Y)$ . In the sequel, we use integer  $x_i$  and vertex  $x_i$  interchangeably, and use an edge of  $G$  and an element of CIP interchangeably.

For a connected component of the bipartite graph  $G(X, Y)$ , let  $X'$  denote its subset of vertices in  $X$  and  $Y'$  denote its subset of vertices in  $Y$ , respectively; then  $X'$  and  $Y'$  are related and they are called a pair of *related sub-multisets* of  $X$  and  $Y$ ; furthermore, the edges in this connected component form a common integer partition for  $X'$  and  $Y'$  and denoted as  $\text{CIP}(X', Y')$ , with  $|\text{CIP}(X', Y')| \geq |X'| + |Y'| - 1$ .

It might happen that the induced bipartite graph  $G(X, Y)$  by any CIP of  $\{X, Y\}$  is connected, or equivalently speaking  $X$  and  $Y$  has no pair of related

proper sub-multisets. In this case  $X$  and  $Y$  are *basic* related multisets. For example,  $X = \{3, 3, 4\}$  and  $Y = \{6, 2, 2\}$  are not basic since  $\{3, 3\}$  and  $\{6\}$  is a pair of related proper sub-multisets; while  $X = \{1, 4\}$  and  $Y = \{2, 3\}$  are basic. Define the size of a pair of related multisets  $X$  and  $Y$  to be the total number of elements in the two multisets, *i.e.*  $|X| + |Y|$ .

**Lemma 2.** [5,6] *If  $X$  and  $Y$  are a pair of basic related multisets, then  $|\text{OPT}(X, Y)| = |X| + |Y| - 1$ .*

*If the minimum size of any pair of related sub-multisets of  $X$  and  $Y$  is  $c$ , then  $|\text{OPT}(X, Y)| \geq \frac{c-1}{c}(|X| + |Y|)$ .*

In the sequel, a set containing a single element is also denoted by the element, when there is no ambiguity, and  $X - X'$  is the set minus/subtraction operation. The next lemma handles size-2 related sub-multisets.

**Lemma 3.** [5,6] *For an instance  $\{X, Y\}$  of 2-MCIP, if  $x_i = y_j$  for some  $x_i \in X$  and  $y_j \in Y$ , then  $x_i \uplus \text{OPT}(X - x_i, Y - y_j)$  is a minimum CIP for  $X$  and  $Y$ , *i.e.*,  $|\text{OPT}(X, Y)| = |\text{OPT}(X - x_i, Y - y_j)| + 1$ .*

## 2.2 The Algorithm

In this section we present a new approximation algorithm, denoted as Apx65, for computing a CIP for the given two related multisets  $X$  and  $Y$ . The running time and worst-case performance analyses are done in the next section. Essentially, algorithm Apx65 extends the set packing idea in the 5/4-approximation algorithm [5,6], to pack the pairs of basic related sub-multisets of sizes 3, 4, and 5. Nonetheless, our set packing process is different from the process in the 5/4-approximation algorithm, and the performance analysis is built on several new properties we uncover between  $\text{OPT}(X, Y)$  and our  $\text{CIP}_{\text{Apx65}}$ .

Let  $Z = X \cap Y$  denote the sub-multiset of common elements of  $X$  and  $Y$ . By Lemma 3 we know that  $\text{OPT}(X - Z, Y - Z) \uplus Z$  is an optimal CIP for  $X$  and  $Y$ . Therefore, in the sequel we assume without loss of generality that  $X$  and  $Y$  do not share any common integer. In the first step of algorithm Apx65, all pairs of basic related sub-multisets of  $X$  and  $Y$  of sizes 3, 4, and 5 are identified. A pair of basic related sub-multisets of size  $i$  is called an  $i$ -set, for  $i = 3, 4, 5$ ; the weight  $w(\cdot)$  of a 3-set (4, 5-set, respectively) is set to 3 (2, 1, respectively). We use  $\mathcal{C}$  to denote this collection of  $i$ -sets for  $i = 3, 4, 5$ .

Let the ground multiset  $U$  contain all elements of  $X$  and  $Y$  that appear in some  $i$ -set of  $\mathcal{C}$ . In the second step, the algorithm is to find a set packing of large weight for the *Weighted Set Packing* [4] instance  $(U, \mathcal{C})$ . To do so, a graph  $H$  is constructed in which a vertex one-to-one corresponds to an  $i$ -set of  $\mathcal{C}$  and two vertices are adjacent if and only if the two corresponding  $i$ -sets intersect. This step of computing a *heavy* set packing is iterative [4], denoted by Greedy37: suppose  $P$  is the current set packing (equivalently an independent set in  $H$ , which was initialized to contain all isolated vertices of  $H$ ), and let  $w^2(P) = \sum_{p \in P} w^2(p)$  be the sum of squared weights of all  $i$ -sets of  $P$ ; an independent set  $T$  of  $H$  (equivalently a sub-collection of disjoint  $i$ -sets of  $\mathcal{C}$ ) *improves*  $w^2(P)$  if

$w^2(T) > w^2(N(T, P))$ , where  $N(T, P)$  denotes the closed neighborhood of  $T$  in  $P$ ; finally, if there is an independent set  $T$  of size  $\leq 37$  which improves  $w^2(P)$ , then  $P$  is replaced by  $(P - N(T, P)) \cup T$ ; otherwise, the process terminates and returns the current  $P$  as the solution set packing.

Input: Related multisets $X$ and $Y$ . Output: A common integer partition $\text{CIP}_{\text{Apx65}}$ of $X$ and $Y$ .
<ol style="list-style-type: none"> <li>1. 1.1. Let <math>Z = X \cap Y</math>;</li> <li style="padding-left: 20px;">1.2. <math>X \leftarrow X - Z, Y \leftarrow Y - Z</math>;</li> <li style="padding-left: 20px;">1.3. Identify <math>\mathcal{C}</math> of all basic related sub-multisets of sizes 3, 4, 5;</li> <li>2. 2.1. Let <math>U</math> be the ground multiset;</li> <li style="padding-left: 20px;">2.2. Compute a heavy set packing <math>P</math> for instance <math>(U, \mathcal{C})</math> by Greedy37;</li> <li>3. 3.1. Let <math>X'</math> and <math>Y'</math> be the sub-multisets of elements covered by <math>P</math>;</li> <li style="padding-left: 20px;">3.2. Run Apx21 to compute <math>\text{CIP}_{\text{Apx21}}(X - X', Y - Y')</math>;</li> <li style="padding-left: 20px;">3.3. Return <math>Z \uplus \left( \biguplus_{X_0 \uplus Y_0 \in P} \text{OPT}(X_0, Y_0) \right) \uplus \text{CIP}_{\text{Apx21}}(X - X', Y - Y')</math>.</li> </ol>

**Fig. 1.** A high-level description of algorithm Apx65

Let  $P$  denote the set packing computed in the second step, and  $X'$  and  $Y'$  denote the sub-multisets of  $X$  and  $Y$ , respectively, of which the elements are “covered” by the  $i$ -sets of  $P$ . Note that  $P$  is maximal, in the sense that no more  $i$ -set of  $\mathcal{C}$  can be appended to  $P$ . Therefore, in the remainder 2-MCIP instance  $(X - X', Y - Y')$ , the minimum size of any pair of related sub-multisets of  $X - X'$  and  $Y - Y'$  is at least 6. In the last step, algorithm Apx21 is run on instance  $(X - X', Y - Y')$  to output a solution  $\text{CIP}_{\text{Apx21}}(X - X', Y - Y')$ ; the final solution  $\text{CIP}_{\text{Apx65}}(X, Y)$  is

$$Z \uplus \left( \biguplus_{X_0 \uplus Y_0 \in P} \text{OPT}(X_0, Y_0) \right) \uplus \text{CIP}_{\text{Apx21}}(X - X', Y - Y'), \quad (1)$$

where  $X_0 \uplus Y_0 \in P$  is an  $i$ -set in the computed set packing  $P$ . A high-level description of algorithm Apx65 is depicted in Fig. 1.

### 2.3 Performance Analysis

The key to the performance guarantee is to analyze the quality of the computed set packing  $P$  in the second step of the algorithm. Let  $P_i$  denote the collection of  $i$ -sets in  $P$ , for  $i = 3, 4, 5$ , respectively. For the weighted set packing instance  $(U, \mathcal{C})$ , we consider one optimal set packing  $Q^*$  and let  $Q_i^*$  denote the sub-collection of  $i$ -sets in  $Q^*$ , for  $i = 3, 4, 5$ , respectively. Let  $p_i = |P_i|$  and  $q_i^* = |Q_i^*|$ , for  $i = 3, 4, 5$ .

We further let  $Q_{ij}^*$  be the sub-collection of  $Q_i^*$ , each  $i$ -set of which intersects with exactly  $j$  sets of the computed set packing  $P$ , for  $i = 3, 4, 5$  and

$j = 1, 2, \dots, i$ . Let  $q_{ij}^* = |Q_{ij}^*|$ . Because the set packing  $P$  is maximal, each set of  $Q^*$  must intersect with certain set(s) in  $P$ . This implies

$$q_i^* = \sum_{j=1}^i q_{ij}^*, \quad i = 3, 4, 5. \tag{2}$$

On the other hand, every  $i$ -set of  $Q_{ij}^*$  intersects with exactly  $j$  sets of  $P$ ; therefore

$$\sum_{i=3}^5 \sum_{j=1}^i (j \times q_{ij}^*) \leq |X'| + |Y'| = \sum_{i=3}^5 (i \times p_i). \tag{3}$$

Eq. (2) and Eq. (3) together give

$$\begin{aligned} & 3q_3^* + 2q_4^* + q_5^* \\ &= 3 \sum_{j=1}^3 q_{3j}^* + 2 \sum_{j=1}^4 q_{4j}^* + \sum_{j=1}^5 q_{5j}^* \\ &\leq \left( \sum_{j=1}^3 j q_{3j}^* + 2q_{31}^* + q_{32}^* \right) + \left( \sum_{j=1}^4 j q_{4j}^* + q_{41}^* \right) + \left( \sum_{j=1}^5 j q_{5j}^* \right) \\ &= \left( \sum_{i=3}^5 \sum_{j=1}^i j q_{ij}^* \right) + 2q_{31}^* + q_{32}^* + q_{41}^* \\ &\leq (3p_3 + 4p_4 + 5p_5) + 2q_{31}^* + q_{32}^* + q_{41}^*. \end{aligned} \tag{4}$$

The following Lemma 4 states a key structural relationship between the computed set packing  $P$  and the optimal set packing  $Q^*$ . Section 3 is devoted to the proof of this lemma.

**Lemma 4.**  $3q_3^* + 2q_4^* + q_5^* \leq 5(p_3 + p_4 + p_5)$ .

By Lemma 3, we assume that there are no common integer elements between the two input multisets  $X$  and  $Y$ . Lemma 5 presents a quality guarantee on the computed  $\text{CIP}_{\text{Apx65}}(X, Y)$ , in terms of the set packing  $P$ .

**Lemma 5.**  $|\text{CIP}_{\text{Apx65}}(X, Y)| \leq m + n - (p_3 + p_4 + p_5 + 1)$ .

*Proof.* Note from the description of algorithm Apx65 in Fig. 1, that for every  $i$ -set of the computed set packing  $P$ , its common integer partition has the minimum size  $i - 1$ , for  $i = 3, 4, 5$ . That is,

$$\left| \bigoplus_{X_0 \uplus Y_0 \in P} \text{OPT}(X_0, Y_0) \right| = 2p_3 + 3p_4 + 4p_5, \tag{5}$$

where  $X_0 \uplus Y_0 \in P$  is an  $i$ -set in the computed set packing  $P$ . On the other hand, on the remainder instance of  $X - X'$  and  $Y - Y'$ , algorithm Apx21 returns a solution

$$|\text{CIP}_{\text{Apx21}}(X - X', Y - Y')| \leq m + n - (3p_3 + 4p_4 + 5p_5) - 1. \tag{6}$$

The lemma immediately follows from Eqs. (5, 6). □

We now estimate  $\text{OPT}(X, Y)$ . Let  $Q'_i$ , for  $i = 3, 4, 5$  be the collection of pairs of basic related multisets of size  $i$  induced by  $\text{OPT}(X, Y)$ , and let  $q'_i = |Q'_i|$ . It is clear that

$$3q'_3 + 2q'_4 + q'_5 \leq 3q_3^* + 2q_4^* + q_5^* \tag{7}$$

because  $Q^*$  is the maximum weight set packing of the instance  $(U, \mathcal{C})$  and certainly  $Q = Q'_3 \cup Q'_4 \cup Q'_5$  is also a set packing.

**Lemma 6.**  $|\text{OPT}(X, Y)| \geq \frac{5}{6}(m + n) - \frac{1}{6}(3q_3^* + 2q_4^* + q_5^*)$ .

*Proof.* Note that for every  $i$ -set of the set packing  $Q$ , its common integer partition has the minimum size  $i - 1$ , for  $i = 3, 4, 5$ . Every other connected component in graph  $G(X, Y)$  induced by  $\text{OPT}(X, Y)$  has size at least 6. Therefore, by Lemma 2 we have

$$\begin{aligned} |\text{OPT}(X, Y)| &\geq 2q'_3 + 3q'_4 + 4q'_5 + \frac{5}{6}(m + n - 3q'_3 - 4q'_4 - 5q'_5) \\ &= \frac{5}{6}(m + n) - \frac{1}{6}(3q'_3 + 2q'_4 + q'_5) \\ &\geq \frac{5}{6}(m + n) - \frac{1}{6}(3q_3^* + 2q_4^* + q_5^*). \end{aligned}$$

This proves the lemma. □

**Theorem 1.** *Algorithm Apx65 is a  $\frac{6}{5}$ -approximation for 2-MCIP.*

*Proof.* We first examine the time complexity of algorithm Apx65. From the description of algorithm Apx65 in Fig 1, steps 1.1 and 1.2 can be done in  $O(m+n)$  and step 1.3 takes  $O((m+n)^5)$  time as there are at most  $O((m+n)^5)$  sets in  $\mathcal{C}$ . Our weighting scheme ensures that each iteration of Greedy37 increases the sum of squared weights by at least 1. Note that the sum of squared weights of any set packing is upper bounded by  $3(m+n)$ . We conclude that the total number of iterations in Greedy37 is  $O(m+n)$ . In each iteration, we check every sub-collection of  $\mathcal{C}$  of size  $\leq 37$ , which takes  $O((m+n)^{5 \times 37}) = O((m+n)^{186})$  time. That is, step 2 costs  $O((m+n)^{186})$  time. Step 3 takes linear time as algorithm Apx21 runs in linear time. In summary, the total running time of our algorithm Apx65 is  $O((m+n)^{186})$ .

For its worst case performance ratio, by Lemmas 4, 5 and 6, we have

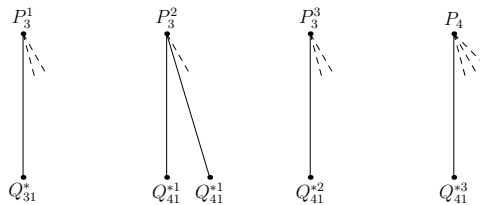
$$\begin{aligned} \frac{\text{CIP}_{\text{Apx65}}(X, Y)}{\text{OPT}(X, Y)} &\leq \frac{m + n - (p_3 + p_4 + p_5 + 1)}{\frac{5}{6}(m + n) - \frac{1}{6}(3q_3^* + 2q_4^* + q_5^*)} \\ &\leq \frac{6}{5} \times \frac{m + n - (p_3 + p_4 + p_5 + 1)}{m + n - \frac{1}{5}(3q_3^* + 2q_4^* + q_5^*)} \\ &\leq \frac{6}{5} \times \frac{m + n - (p_3 + p_4 + p_5 + 1)}{m + n - (p_3 + p_4 + p_5)} \\ &< \frac{6}{5}, \end{aligned}$$

where  $m = |X|$  and  $n = |Y|$ . Therefore, Apx65 is a  $\frac{6}{5}$ -approximation. □

### 3 Proof of Lemma 4

Recall the termination condition of algorithm Greedy37 for computing the heavy set packing  $P$ , that is, there is no independent set  $T$  such that  $|T| \leq 37$  and  $T$  improves  $w^2(P)$ . Also recall the weighting scheme  $(w_3, w_4, w_5) = (3, 2, 1)$ , where  $w_i$  is the weight of an  $i$ -set of  $\mathcal{C}$ . We summarize in the following Lemma 7 some useful properties of the sets in  $Q_{31}^*$ ,  $Q_{41}^*$ , and  $Q_{32}^*$ , see also Fig. 2. Their proofs are straightforward using the termination condition and the weight scheme, and we skip them.

- Lemma 7.** (a) *Every set of  $Q_{31}^*$  intersects with a set of  $P_3$ , and no other set of  $Q_{31}^* \cup Q_{41}^*$  can intersect with this set of  $P_3$ ; such sets of  $P_3$  form a sub-collection denoted as  $P_3^1$ .*
- (b) *Every set of  $Q_{41}^*$  intersects with a set of  $P_3 \cup P_4$ .*
- (b1) *If two sets of  $Q_{41}^*$  intersect with a common set of  $P$ , then this set belongs to  $P_3$ , and no other set of  $Q_{41}^*$  can intersect with this set of  $P_3$ ; such sets of  $P_3$  form a sub-collection denoted as  $P_3^2$ , and such sets of  $Q_{41}^*$  form a sub-collection denoted as  $Q_{41}^{*1}$ .*
- (b2) *If only one set of  $Q_{41}^*$  intersects with a set of  $P_3$ , then no other set of  $Q_{31}^* \cup Q_{41}^*$  can intersect with this set of  $P_3$ ; such sets of  $P_3$  form a sub-collection denoted as  $P_3^3$ , and such sets of  $Q_{41}^*$  form a sub-collection denoted as  $Q_{41}^{*2}$ .*
- (b3) *Otherwise, a set of  $Q_{41}^*$  intersects with a set of  $P_4$ , and no other set of  $Q_{31}^* \cup Q_{41}^*$  can intersect with this set of  $P_4$ ; such sets of  $Q_{41}^*$  form a sub-collection denoted as  $Q_{41}^{*3}$ .*
- Let  $P_3^4 = P_3 - P_3^1 - P_3^2 - P_3^3$ . Clearly,  $\{P_3^1, P_3^2, P_3^3, P_3^4\}$  is a partition of  $P_3$ ; so is  $\{Q_{41}^{*1}, Q_{41}^{*2}, Q_{41}^{*3}\}$  a partition of  $Q_{41}^*$ .
- (c) *Every set of  $Q_{32}^*$  must intersect a set of  $P_3$ .*



**Fig. 2.** The definitions of sub-collections of  $P_3$  and  $Q_{41}^*$  using the set intersecting configurations, where a solid (dashed, respectively) line indicates a firm (possible, respectively) set intersection

Let  $p_3^j = |P_3^j|$  for  $j = 1, 2, 3, 4$ , and  $q_{41}^{*j} = |Q_{41}^{*j}|$  for  $j = 1, 2, 3$ .

**Lemma 8.** *We have the following relationships:  $p_3 = p_3^1 + p_3^2 + p_3^3 + p_3^4$ ,  $q_{41}^* = q_{41}^{*1} + q_{41}^{*2} + q_{41}^{*3}$ ,  $p_3^1 = q_{31}^*$ ,  $p_3^2 = \frac{1}{2}q_{41}^{*1}$ , and  $p_3^3 = q_{41}^{*2}$ .*

*Proof.* The first two equalities hold since, by Lemma 7(b),  $\{P_3^1, P_3^2, P_3^3, P_3^4\}$  is a partition of  $P_3$ , and  $\{Q_{41}^{*1}, Q_{41}^{*2}, Q_{41}^{*3}\}$  is a partition of  $Q_{41}^*$ .

The third equality holds by Lemma 7(a) that the sets of  $Q_{31}^*$  and the sets of  $P_3^1$  one-to-one correspond to each other. Analogously, the fourth equality holds due to Lemma 7(b1) that the sets of  $Q_{41}^{*1}$  are paired up, and these pairs and the sets of  $P_3^2$  one-to-one correspond to each other; the fifth equality holds by Lemma 7(b2) that the sets of  $Q_{41}^{*2}$  and the sets of  $P_3^3$  one-to-one correspond to each other.  $\square$

We next construct a bipartite graph  $H'$ , which is an induced subgraph of graph  $H$  that we constructed for the Weighted Set Packing instance  $(U, \mathcal{C})$ , as follows: One subset of vertices of  $H'$  is  $Q_{31}^* \cup Q_{32}^* \cup Q_{41}^*$  (which is a sub-collection of the optimal set packing  $Q^*$ ), and the other subset of vertices of  $H'$  is  $P$  (which is the computed set packing), and again two vertices are adjacent if and only if the corresponding two sets intersect. In the sequel, we use the set of  $\mathcal{C}$  and the vertex of graph  $H$  (or  $H'$ ) interchangeably; we also abuse the sub-collection, such as  $Q_{31}^*$ , of sets to denote the corresponding vertex subset in graph  $H$  (or  $H'$ ). Once again recall that the termination condition of Greedy37 tells that there is no improving subset of  $Q_{31}^* \cup Q_{32}^* \cup Q_{41}^*$  of size  $\leq 37$ .

We prove Lemma 4 by showing that the inequality holds in every connected component of graph  $H'$ , followed by a straightforward linear summation over all connected components. We therefore assume without loss of generality that graph  $H'$  is connected.

**Lemma 9.** *If  $p_4 + p_5 \geq 2$ , then in graph  $H'$  the length of the shortest path between any two vertices  $a, b \in P_4 \cup P_5$  is  $d(a, b) \geq 76$ ; consequently,  $p_3 \geq 37$  and  $p_4 + p_5 \leq \frac{1}{18}p_3$ .*

*Proof.* Let  $a$  and  $b$  be two vertices of  $P_4 \cup P_5$  such that there is no other vertex from  $P_4 \cup P_5$  on a shortest path connecting them in graph  $H'$ . Since  $H'$  is bipartite, this path has an even length and is denoted as

$$\langle a = a_0, c_0, a_1, c_1, \dots, a_\ell, c_\ell, a_{\ell+1} = b \rangle,$$

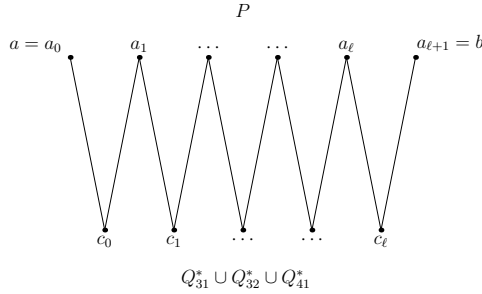
for some  $\ell \geq 0$  (see Fig. 3). Since every vertex  $c_i$  on the path has degree at least 2, it has to belong to  $Q_{32}^*$  and consequently it has degree exactly 2 in graph  $H'$ . It follows that for the independent set  $T = \{c_0, c_1, \dots, c_\ell\}$ ,  $w^2(T) = 9(\ell + 1)$  and  $w^2(N(T, P)) = w^2(\{a_0, a_1, \dots, a_{\ell+1}\}) \leq 9\ell + 8$ . We conclude that  $\ell \geq 37$  as otherwise  $T$  would be an improving subset of vertices. Therefore, the length of the above shortest path is  $d(a, b) \geq 76$  and it contains at least  $\ell \geq 37$  vertices of  $P_3$ .

To prove the second half of the lemma, we notice that graph  $H'$  is connected. For every vertex  $a \in P_4 \cup P_5$ , we pick arbitrarily another vertex  $b \in P_4 \cup P_5$  and consider a shortest path connecting them in graph  $H'$  that does not contain any other vertex from  $P_4 \cup P_5$ :

$$\langle a = a_0, c_0, a_1, c_1, \dots, a_\ell, c_\ell, a_{\ell+1} = b \rangle,$$

for some  $\ell \geq 37$ . Initially every vertex of  $P_4 \cup P_5$  is worth 1 token; through this path, vertex  $a$  distributes its 1 token evenly to vertices  $a_1, a_2, \dots, a_{18}$ , which are





**Fig. 3.** The configuration of the shortest path connecting  $a, b \in P_4 \cup P_5$

all vertices of  $P_3$ . After every vertex of  $P_4 \cup P_5$  has distributed its token, from  $\ell \geq 37$  we conclude that every vertex of  $P_3$  receives no more than  $\frac{1}{18}$  token. Therefore,

$$p_4 + p_5 \leq \frac{1}{18}p_3.$$

This proves the lemma. □

From Lemma 9, we see that the number of 4-sets and 5-sets in the computed set packing  $P$  is very small compared against the number of 3-sets. In the following Lemma 10 we prove that  $2q_{31}^* + q_{32}^* + q_{41}^* \leq 2p_3 + p_4$  through an amortized analysis. By Eq. (4), Lemma 10 is sufficient to prove Lemma 4.

**Lemma 10.**  $2q_{31}^* + q_{32}^* + q_{41}^* \leq 2p_3 + p_4$ .

*Proof.* The proof of the lemma is through an amortized analysis, and is done via five distinct cases. We assign 2 tokens for each vertex of  $Q_{31}^*$  and 1 token for each vertex of  $Q_{32}^* \cup Q_{41}^*$ . So we have a total of  $2q_{31}^* + q_{32}^* + q_{41}^*$  tokens. We will prove that  $2q_{31}^* + q_{32}^* + q_{41}^* \leq 2p_3 + p_4$  by distributing these tokens to the vertices of  $P$ .

We consider the following five distinct cases of graph  $H'$ , which are separately dealt:

- Case 1.  $q_{31}^* = q_{41}^* = 0$ ,
- Case 2.  $q_{31}^* = 1$  and  $q_{41}^* = 0$ ,
- Case 3.  $q_{31}^* = 0$  and  $q_{41}^* = 1$ ,
- Case 4.  $q_{31}^* = 0$  and  $q_{41}^* = 2$  with either  $q_{41}^{*1} = 2$  or  $q_{41}^{*2} = 2$ ,
- Case 5.  $q_{31}^* + q_{41}^* \geq 2$  excluding Case 4.

Due to space constraint, in the following we only prove the inequality for Case 1: if  $q_{31}^* = q_{41}^* = 0$ , then  $2q_{31}^* + q_{32}^* + q_{41}^* \leq 2p_3 + p_4$ .

Firstly, from Lemma 7(c), every set of  $Q_{32}^*$  must intersect a set of  $P_3$ . If  $p_4 + p_5 \leq 1$ , then we have  $2q_{32}^* \leq 3p_3 + 5$ . It follows that when  $p_3 \geq 5$ ,  $2q_{32}^* \leq 3p_3 + 5 \leq 4p_3$  and thus  $2q_{31}^* + q_{32}^* + q_{41}^* \leq 2p_3 + p_4$ . When  $p_3 = 4$  (3, 2, 1, 0, respectively),  $w^2(P) \leq 40$  (31, 22, 13, 4, respectively) and thus  $q_{32}^* \leq 4$  (3, 2, 1, 0,

respectively) by algorithm Greedy37; that is,  $q_{32}^* \leq p_3$ , and consequently  $2q_{31}^* + q_{32}^* + q_{41}^* \leq 2p_3 + p_4$ .

If  $p_4 + p_5 > 1$ , every set of  $Q_{32}^*$  distributes  $\frac{1}{2}$  token to each adjacent set of  $P$ . Note that every  $i$ -set of  $P$  receives at most  $\frac{i}{2}$  token, by Lemma 9,

$$q_{32}^* \leq \sum_{i=3}^5 \frac{i}{2} p_i \leq \frac{3}{2} p_3 + \frac{5}{2} (p_4 + p_5) \leq \frac{3}{2} p_3 + \frac{5}{2} \times \frac{1}{18} p_3 = \frac{59}{36} p_3,$$

and consequently  $2q_{31}^* + q_{32}^* + q_{41}^* \leq 2p_3 + p_4$ .

The other four cases can be analogously proved. □

### 4 A 0.6k-Approximation Algorithm for k-MCIP

Given an instance of the  $k$ -MCIP problem  $\{X_1, X_2, \dots, X_k\}$ , we first divide these  $k$  multisets into  $\lfloor k/2 \rfloor$  pairs  $\{X_{2i-1}, X_{2i}\}$ ,  $i = 1, 2, \dots, \lfloor k/2 \rfloor$ , plus the last multiset  $X_k$  if  $k$  is odd. Next, we run algorithm Apx65 on each pair  $\{X_{2i-1}, X_{2i}\}$  to obtain a solution  $Z_i = \text{CIP}_{\text{Apx65}}(X_{2i-1}, X_{2i})$ , for  $i = 1, 2, \dots, \lfloor k/2 \rfloor$ , plus  $Z_{(k+1)/2} = X_k$  if  $k$  is odd. We continue this dividing and running Apx65 on  $\{Z_1, Z_2, \dots, Z_{\lceil (k+1)/2 \rceil}\}$  if  $\lceil (k+1)/2 \rceil \geq 2$ , and repeat until we have only one multiset left, denoted as  $\text{CIP}_{\text{final}}$ . Clearly,  $\text{CIP}_{\text{final}}$  is a common integer partition of the given multisets  $X_1, X_2, \dots, X_k$ .

**Theorem 2.** *k-MCIP admits a 0.6k-approximation algorithm when k is even, or a (0.6k + 0.4)-approximation algorithm when k is odd.*

*Proof.* The algorithm in the last paragraph producing a feasible solution  $\text{CIP}_{\text{final}}$  runs in polynomial time. We next estimate its performance, and assume that  $k$  is even. By Theorem 1, we have  $|Z_i| < \frac{6}{5} |\text{OPT}(X_{2i-1}, X_{2i})|$ , for  $i = 1, 2, \dots, k/2$ . Let  $\text{OPT}$  denote the minimum common integer partition for  $X_1, X_2, \dots, X_k$ . One clearly sees that  $|\text{OPT}(X_{2i-1}, X_{2i})| \leq |\text{OPT}|$ , and from Lemma 1 we have

$$|\text{CIP}_{\text{final}}| < \sum_{i=1}^{k/2} |Z_i| < \sum_{i=1}^{k/2} \frac{6}{5} |\text{OPT}| = \frac{3k}{5} |\text{OPT}|.$$

When  $k$  is odd,

$$|\text{CIP}_{\text{final}}| < \sum_{i=1}^{(k-1)/2} |Z_i| + |X_k| < \sum_{i=1}^{(k-1)/2} \frac{6}{5} |\text{OPT}| + |\text{OPT}| = \frac{3k+2}{5} |\text{OPT}|,$$

using  $|X_k| \leq |\text{OPT}|$  from Lemma 1. This completes the proof. □

### 5 Conclusion

We presented an improved  $\frac{6}{5}$ -approximation algorithm for the 2-MCIP problem; the previous best approximation algorithm has a performance ratio of  $\frac{5}{4}$  and

was designed in 2006. Subsequently, we obtained an absolute  $0.6k$ -approximation algorithm for  $k$ -MCIP when  $k$  is even (when  $k$  is odd, the approximation ratio is  $0.6k+0.4$ ). It is worth pointing out that the ratio of  $0.5625k$  in [11] is asymptotic, that it holds for only sufficiently large  $k$ ; while our ratio of  $0.6k$  is absolute, that it holds for all  $k$ .

**Acknowledgments.** This research was supported in part by NSERC and a visiting professorship from Zhejiang Sci-Tech University awarded to GL.

## References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithm, and Applications. China Machine Press (2005)
2. Andrews, G.: The Theory of Partitions. Addison-Wesley (1976)
3. Andrews, G., Eriksson, K.: The Integer Partitions. Cambridge University Press (2004)
4. Berman, P.: A  $d/2$  approximation for maximum weight independent set in  $d$ -claw free graphs. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 214–219. Springer, Heidelberg (2000)
5. Chen, X., Liu, L., Liu, Z., Jiang, T.: On the minimum common integer partition problem. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) CIAC 2006. LNCS, vol. 3998, pp. 236–247. Springer, Heidelberg (2006)
6. Chen, X., Liu, L., Liu, Z., Jiang, T.: On the minimum common integer partition problem. ACM Transactions on Algorithms **5**, Article 12 (2008)
7. Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T.: The assignment of orthologous genes via genome rearrangement. IEEE/ACM Transactions on Computational Biology and Bioinformatics **2**, 302–315 (2005)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press, Cambridge (2001)
9. Kann, V.: Maximum bounded 3-dimensional matching is MAX SNP-complete. Information Processing Letters **37**, 27–35 (1991)
10. Woodruff, D.P.: Better approximations for the minimum common integer partition problem. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 248–259. Springer, Heidelberg (2006)
11. Zhao, W., Zhang, P., Jiang, T.: A network flow approach to the minimum common integer partition problem. Theoretical Computer Science **369**, 456–462 (2006)

# Positive Semidefinite Relaxation and Approximation Algorithm for Triple Patterning Lithography

Tomomi Matsui<sup>1</sup>(✉), Yukihide Kohira<sup>2</sup>, Chikaaki Kodama<sup>3</sup>,  
and Atsushi Takahashi<sup>1</sup>

<sup>1</sup> Tokyo Institute of Technology, Tokyo, Japan  
matsui.t.af@m.titech.ac.jp

<sup>2</sup> The University of Aizu, Aizu-Wakamatsu, Fukushima, Japan

<sup>3</sup> Toshiba Corporation, Yokohama, Kanagawa, Japan

**Abstract.** Triple patterning lithography (TPL) is one of the major techniques for 14 nm technology node and beyond. This paper discusses TPL layout decomposition which maximizes objective value representing decomposition quality. We introduce a maximization problem of the weighted sum of resolved conflicts and unused stitch candidates. We propose a polynomial time  $(7/9)$ -approximation algorithm based on positive semidefinite relaxation and randomized rounding procedure.

Our algorithm returns a decomposition such that the expectation of the corresponding objective value is at least  $(7/9)$  times the optimal value even in the worst case problem instance. To our knowledge, the result is the first approximation algorithm with a constant approximation ratio for TPL.

**Keywords:** Triple patterning · Positive semidefinite relaxation · Approximation algorithm

## 1 Introduction

As the feature size decreases, various types of techniques including design for manufacture are investigated in lithography. Although extreme ultra violet (EUV) and electric beam lithography (EBL) are used as the next generation lithography recently, they are not widely used due to their slow throughput and etc. Triple patterning lithography (TPL) and self-aligned double patterning (SADP) are candidates for the 14 nm node. Among them, TPL receives more attention from industry to fabricate complicated patterns which are typically found in metal 1 layer since SADP restricts the flexibility of pattern. For example, researches on TPL layout decomposition [2, 5, 9–12, 14, 15] and TPL aware design [6, 7, 13] are found.

The problem of finding an assignment of polygons in TPL is essentially equivalent to a problem of vertex 3-coloring for undirected graph, which is NP-hard in general [3, 14]. Therefore, methods proposed so far are heuristics, or exact solvers such as ILP and SAT are used for small problems.

As for TPL layout decomposition, Fang et al. [2] proposed several graph simplification techniques. Kuang and Young [5] and Zhang et al. [15] proposed several coloring heuristics. Several structural restrictions were discussed by Tian et al. in [9, 10] and SAT solver was used. Yu et al. [14] proposed a decomposition method using a vector programming with positive semidefinite programming (SDP) techniques, and the density balance was taken into account in [12]. Also, Yu et al. [11] proposed a method using ILP for TPL with cut process where third mask is used to trim patterns. Although various methods have been proposed so far, to our knowledge, there is no method that has theoretical guarantees except the cases when exact solvers can output the results.

In this paper, TPL layout decomposition which maximizes objective value representing decomposition quality is discussed. We introduce a maximization problem of the weighted sum of resolved conflicts and unused stitch candidates. Our maximization problem is essentially equivalent to an ordinary problem of minimizing weighted sum of the number of conflicts and number of used stitch candidates [2, 5, 6, 14, 15]. We propose a polynomial time  $(7/9)$ -approximation algorithm based on positive semidefinite relaxation and randomized rounding procedure. Our algorithm returns a decomposition such that the expectation of the corresponding objective value is at least  $(7/9)$  times the optimal value even in the worst case problem instance. To our knowledge, the result is the first approximation algorithm with a constant approximation ratio for TPL.

Our positive semidefinite relaxation, which gives a maximization problem, is essentially equivalent to that proposed in [14], which minimizes an objective function. It is well-known that a positive semidefinite programming problem can be solved by interior point methods in polynomial time. The runtime of SDP solver required in our proposed algorithm is expected to be comparable to the methods in [12, 14] since the formulations are similar. However, our proposed algorithm has the guarantee in the quality of the obtained solution. One can also apply our randomized rounding procedure to an optimal solution of the positive semidefinite relaxation problems proposed in [12, 14].

The rest of this paper is organized as follows: in Section 2, we formulate a problem of TPL layout decomposition. A vector programming formulation is described in Section 3. Section 4 presents a positive semidefinite relaxation problem. We propose a randomized rounding procedure in Section 5, followed by a discussion of an approximation ratio in Section 6. Finally, we discuss the synthesis of our rounding procedure and existing methods, and conclude in Section 7.

## 2 Problem

Let  $V = \{1, 2, \dots, n\}$  be a set of polygons. A polygon may represent a polygon decomposed by given stitch candidates. (See Figure 1(a).) In the rest of this paper, a polygon  $v \in V$  is also called a “*vertex*.” We introduce a simple graph (containing no loops or multiple edges) defined on vertex set  $V$ . (See Figure 1(b).) A stitch edge is defined between two polygons if and only if two polygons are decomposed by a stitch candidate. A polygon conflict edge is defined between

two polygons if and only if two polygons are too close to assign the same mask. A set of conflict edges and a set of stitch edges are denoted by  $E_C$  and  $E_S$ , respectively. Here we note that both  $E_C$  and  $E_S$  are families of unordered pairs of polygons in  $V$ . (See Figure 1(c).)

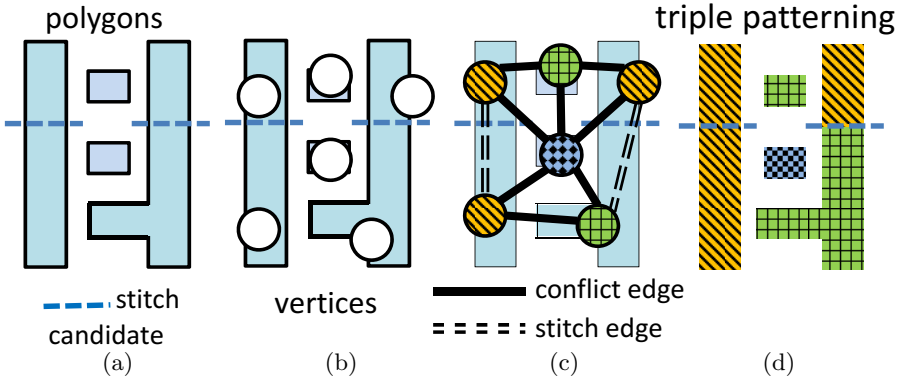


Fig. 1. Set of polygons and a graph

This paper deals with a problem to assign one of three masks to each polygon. The problem of finding an assignment of polygons is essentially equivalent to a vertex three coloring problem. We represent a 3-coloring of polygons  $V$  by a map  $c : V \rightarrow \{0, 1, 2\}$ . (See Figures 1(c) and (d).) When  $c(v) = i$ , we say that vertex  $v$  has color  $i$ . Given a 3-coloring  $c$ , a subset of resolved conflict edges, defined by

$$\overline{CE}(c) \stackrel{\text{def.}}{=} \{\{u, v\} \in E_C \mid c(u) \neq c(v)\},$$

denotes a subset of conflict edges connecting vertices whose colors are different. Similarly, we define a set of unused stitch candidates

$$\overline{SE}(c) \stackrel{\text{def.}}{=} \{\{u, v\} \in E_S \mid c(u) = c(v)\},$$

i.e., a subset of stitch edges connecting vertices with a common color.

This paper discusses TPL layout decomposition which maximizes objective value representing decomposition quality. The problem maximizes the weighted sum of resolved conflicts and unused stitch candidates;

$$\begin{aligned} \text{P1: maximize} \quad & \alpha_1 |\overline{CE}(c)| + \alpha_2 |\overline{SE}(c)| \\ \text{subject to} \quad & c(v) \in \{0, 1, 2\} \quad (\forall v \in V). \end{aligned}$$

The above problem is similar to the correlation clustering problem introduced by Bansal, Blum, and Chawla [1] and an approximation algorithm is proposed by Swamy [8].

### 3 Vector Programming

For any 3-coloring  $c$ , we introduce a map, presented in [14], defined by

$$\mathbf{x}(v) \stackrel{\text{def.}}{=} \begin{cases} (1, 0)^\top & (c(v) = 0), \\ \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right)^\top & (c(v) = 1), \\ \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)^\top & (c(v) = 2). \end{cases}$$

The above definition directly implies that  $\left[ c(u) \neq c(v) \leftrightarrow \mathbf{x}(u)^\top \mathbf{x}(v) = -\frac{1}{2} \right]$  and  $\left[ c(u) = c(v) \leftrightarrow \mathbf{x}(u)^\top \mathbf{x}(v) = 1 \right]$ . Then the sizes of  $\overline{\text{CE}}(c)$  and  $\overline{\text{SE}}(c)$  satisfy

$$|\overline{\text{CE}}(c)| = \sum_{\{u,v\} \in E_C} \left( \frac{-2\mathbf{x}(u)^\top \mathbf{x}(v)}{3} + \frac{2}{3} \right), \text{ and } |\overline{\text{SE}}(c)| = \sum_{\{u,v\} \in E_S} \left( \frac{2\mathbf{x}(u)^\top \mathbf{x}(v)}{3} + \frac{1}{3} \right).$$

From the above properties, we formulate P1 as a vector programming problem;

$$\begin{aligned} \text{P2: max. } & \alpha_1 \sum_{\{u,v\} \in E_C} \left( \frac{-2\mathbf{x}(u)^\top \mathbf{x}(v)}{3} + \frac{2}{3} \right) + \alpha_2 \sum_{\{u,v\} \in E_S} \left( \frac{2\mathbf{x}(u)^\top \mathbf{x}(v)}{3} + \frac{1}{3} \right) \\ \text{s. t. } & \mathbf{x}(v)^\top \in \left\{ (1, 0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right), \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right) \right\}. \end{aligned}$$

### 4 SDP Relaxation

We introduce a matrix  $X$  of variables whose  $(u, v)$  element satisfies  $X_{uv} = \mathbf{x}(u)^\top \mathbf{x}(v)$ . Clearly, diagonal elements of  $X$  are equal to 1, every element of  $X$  is greater than or equal to  $-1/2$  and  $X$  is positive-semidefinite. In the following,  $\mathcal{S}_+^n$  denotes the set of all the symmetric positive-semidefinite  $n \times n$  matrices. Then the problem P2 is formulated as;

$$\begin{aligned} \text{P3: max. } & \alpha_1 \sum_{\{u,v\} \in E_C} \left( \frac{-2X_{uv}}{3} + \frac{2}{3} \right) + \alpha_2 \sum_{\{u,v\} \in E_S} \left( \frac{2X_{uv}}{3} + \frac{1}{3} \right) \\ \text{s. t. } & X_{uv} \geq -\frac{1}{2} && (\forall (u, v) \in V^2), \\ & X_{vv} = 1 && (\forall v \in V), \\ & X \in \mathcal{S}_+^n, \\ & X_{uv} = \mathbf{x}(u)^\top \mathbf{x}(v) && (\forall (u, v) \in V^2), \tag{1} \\ & \mathbf{x}(v)^\top \in \left\{ (1, 0), \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right), \left(-\frac{1}{2}, -\frac{\sqrt{3}}{2}\right) \right\}. \tag{2} \end{aligned}$$

When we remove constraints (1,2) and eliminate variables  $\mathbf{x}(v)$  ( $v \in V$ ), we obtain the following relaxation problem:

$$\begin{aligned} \text{SDP: max.} \quad & \alpha_1 \sum_{\{u,v\} \in E_C} \left( \frac{-2X_{uv}}{3} + \frac{2}{3} \right) + \alpha_2 \sum_{\{u,v\} \in E_S} \left( \frac{2X_{uv}}{3} + \frac{1}{3} \right) \\ \text{s. t.} \quad & X_{uv} \geq -\frac{1}{2} \quad (\forall (u,v) \in V^2), \\ & X_{vv} = 1 \quad (\forall v \in V), \\ & X \in \mathcal{S}_+^n, \end{aligned}$$

which is a positive semidefinite programming problem. Interior point methods solve a positive semidefinite programming problem in polynomial time. If we denote the optimal value of a problem by  $z(\cdot)$ , then it is obvious that  $z(\text{P1}) = z(\text{P2}) = z(\text{P3}) \leq z(\text{SDP})$ .

In the following, we briefly show that our positive semidefinite relaxation is essentially equivalent to that proposed in [14]. It is easy to see that we use the same constraints. The objective of problem SDP is equivalent to minimizing

$$\begin{aligned} & -\alpha_1 \sum_{\{u,v\} \in E_C} \left( \frac{-2X_{uv}}{3} + \frac{2}{3} \right) - \alpha_2 \sum_{\{u,v\} \in E_S} \left( \frac{2X_{uv}}{3} + \frac{1}{3} \right) \\ & = \alpha_1 \sum_{\{u,v\} \in E_C} \left( \frac{2X_{uv}}{3} + \frac{1}{3} \right) + \alpha_2 \sum_{\{u,v\} \in E_S} \left( -\frac{2X_{uv}}{3} + \frac{2}{3} \right) \\ & \quad - |E_C| - |E_S|. \end{aligned}$$

By removing the constant term  $-|E_C| - |E_S|$ , we obtain the objective function of positive semidefinite relaxation problem proposed in [14].

## 5 Randomized Rounding

In this section, we propose a randomized rounding procedure based on the hyperplane separation technique proposed by Goemans and Williamson [4], which outputs a 3-coloring from an optimal solution of the problem SDP defined in the previous section and Cholesky decomposition. For any positive semidefinite symmetric matrix  $X \in \mathcal{S}_+^n$ , there exists a matrix  $M$  satisfying  $X = M^\top M$ . This decomposition is called Cholesky decomposition.

We solve problem SDP and obtain an optimal solution  $\tilde{X}$ . Let  $\tilde{M}^\top \tilde{M}$  be the Cholesky decomposition of  $\tilde{X}$  and  $d$  be the number of rows of  $\tilde{M}$ . Here we note that columns of  $\tilde{M}$  are indexed by polygons in  $V$  and the length of every column vector is equal to 1. For each polygon  $v \in V$ , vector  $\tilde{\mathbf{x}}(v) \in \mathbb{R}^d$  denotes the corresponding column vector of  $\tilde{M}$ . Now we propose a procedure for generating a 3-coloring  $\tilde{c}: V \rightarrow \{0, 1, 2\}$  from a set of (column) vectors  $\tilde{\mathbf{x}}(v) \in \mathbb{R}^d$  ( $v \in V$ ).



**Algorithm RR**

**Step 1:** Generate two random unit vectors  $\mathbf{u}^0, \mathbf{u}^1 \in \mathbb{R}^d$  (satisfying  $\|\mathbf{u}^0\| = \|\mathbf{u}^1\| = 1$ ).

**Step 2:** Construct four vertex subsets

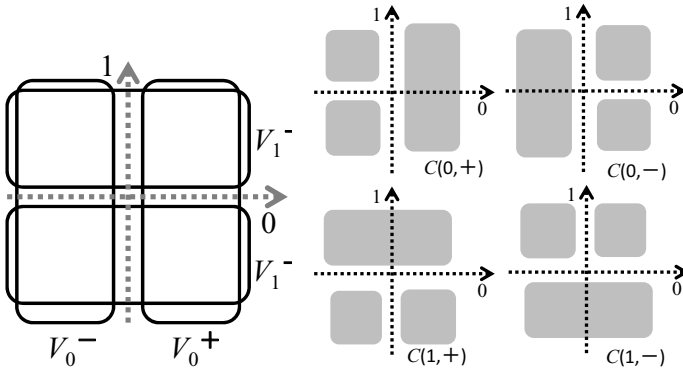
$$V_0^+ = \{v \in V \mid \tilde{\mathbf{x}}(v)^\top \mathbf{u}^0 > 0\}, \quad V_0^- = V \setminus V_0^+,$$

$$V_1^+ = \{v \in V \mid \tilde{\mathbf{x}}(v)^\top \mathbf{u}^1 > 0\}, \quad V_1^- = V \setminus V_1^+.$$

**Step 3:** For each pair of indices  $(i, s) \in \{0, 1\} \times \{+, -\}$ , construct a 3-coloring  $c(i, s) : V \rightarrow \{0, 1, 2\}$ , illustrated in Figure 2, defined by

$$c(i, s)(v) = \begin{cases} 0 & (v \in V_i^s), \\ 1 & (v \in (V \setminus V_i^s) \cap V_{1-i}^+), \\ 2 & (v \in (V \setminus V_i^s) \cap V_{1-i}^-). \end{cases}$$

Output  $\tilde{c} \in \{c(0, +), c(0, -), c(1, +), c(1, -)\}$  which attains the largest objective function value.



**Fig. 2.** Generated two cuts and four 3-colorings

The above procedure finds a TPL layout decomposition, which is a feasible solution of problem P1. In the next section, we discuss the quality of the obtained solution by estimating  $\alpha_1|\overline{\text{CE}}(\tilde{c})| + \alpha_2|\overline{\text{SE}}(\tilde{c})|$ , which corresponds to the value of the objective function in original problem P1.

## 6 Approximation Ratio

In this section, we theoretically estimate the quality of a solution obtained by Algorithm RR. Let  $\tilde{Z}$  be the objective value  $\alpha_1|\overline{\text{CE}}(\tilde{c})| + \alpha_2|\overline{\text{SE}}(\tilde{c})|$  corresponding to 3-coloring  $\tilde{c}$  obtained by Algorithm RR. Here we note that Algorithm RR is

a randomized algorithm, and thus the objective value  $\tilde{Z}$  is a random variable. In the following, we discuss the expectation  $E[\tilde{Z}]$  of the random variable  $\tilde{Z}$ .

We denote the objective values of four 3-colorings generated in Step 3 of Algorithm RR by  $Z(i, s)$  ( $(i, s) \in \{0, 1\} \times \{+, -\}$ ). These four values are also random variables. Then,  $E[\tilde{Z}]$  has the following trivial lower bound:

$$E[\tilde{Z}] = E\left[\max\{Z(0, +), Z(0, -), Z(1, +), Z(1, -)\}\right] \geq E\left[\frac{Z(0, +) + Z(0, -) + Z(1, +) + Z(1, -)}{4}\right].$$

Next, we introduce three families of pairs of polygons appearing in Figure 3, precisely defined as follows:

$$E_0 = \{\{u, v\} \subseteq V \mid u \neq v \text{ and } \exists(i, s) \in \{0, 1\} \times \{+, -\}, \{u, v\} \subseteq V_i^s\},$$

$$E_1 = \left\{\{u, v\} \subseteq V \mid \begin{array}{l} u \neq v \text{ and } \exists(i, s) \in \{0, 1\} \times \{+, -\}, \\ u \in V_i^s \cap V_{1-i}^+ \text{ and } v \in V_i^s \cap V_{1-i}^- \end{array} \right\},$$

$$E_2 = \{\{u, v\} \subseteq V \mid u \neq v\} \setminus (E_0 \cup E_1).$$

Clearly,  $\{E_0, E_1, E_2\}$  is a partition of all the unordered pairs of polygons.

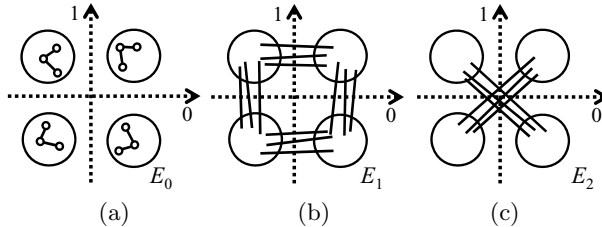


Fig. 3. Three edge subsets

The expectation

$$E\left[\frac{Z(0, +) + Z(0, -) + Z(1, +) + Z(1, -)}{4}\right]$$

is equal to the expectation of the objective value when we choose one of four 3-colorings  $c(0, +), c(0, -), c(1, +), c(1, -)$  uniformly at random. If we choose one of four 3-colorings randomly, then the following properties hold;

1.  $\forall\{u, v\} \in E_0$ , polygons  $u$  and  $v$  have a same color,
2.  $\forall\{u, v\} \in E_1$ , color of  $u$  differs from that of  $v$  with probability  $(3/4)$ ,
3.  $\forall\{u, v\} \in E_2$ , colors of polygons  $u$  and  $v$  are different.

The above properties directly imply that

$$\begin{aligned}
 \mathbb{E} \left[ \tilde{Z} \right] &\geq \mathbb{E} \left[ \frac{Z(0,+) + Z(0,-) + Z(1,+) + Z(1,-)}{4} \right] \\
 &= \mathbb{E} \left[ \begin{aligned} &\alpha_1 \left( |E_2 \cap E_C| + (3/4)|E_1 \cap E_C| \right) \\ &+ \alpha_2 \left( |E_0 \cap E_S| + (1/4)|E_1 \cap E_S| \right) \end{aligned} \right] \\
 &= \alpha_1 \sum_{e \in E_C} \left( \Pr[e \in E_2] + (3/4)\Pr[e \in E_1] \right) \\
 &\quad + \alpha_2 \sum_{e \in E_S} \left( \Pr[e \in E_0] + (1/4)\Pr[e \in E_1] \right).
 \end{aligned}$$

For any pair of polygons  $(u, v) \in V^2$ , we denote the angle of two vectors  $\tilde{\mathbf{x}}(u), \tilde{\mathbf{x}}(v) \in \mathbb{R}^d$  by  $\tilde{\theta}_{uv} = \tilde{\theta}_{vu}[\text{rad}]$ . Then it is obvious that the equalities

$$\cos \tilde{\theta}_{uv} = \|\tilde{\mathbf{x}}(u)\| \cdot \|\tilde{\mathbf{x}}(v)\| \cos \tilde{\theta}_{uv} = \tilde{\mathbf{x}}(u)^\top \tilde{\mathbf{x}}(v) = \tilde{X}_{uv}$$

hold. Since an optimal solution  $\tilde{X}$  of P3 satisfies  $\tilde{X}_{uv} \geq -1/2$  for any  $(u, v) \in V^2$ , every pair  $(u, v) \in V^2$  satisfies inequalities  $0 \leq \tilde{\theta}_{uv} \leq 2\pi/3$ .

For any pair of vectors  $\tilde{\mathbf{x}}(u), \tilde{\mathbf{x}}(v) \in \mathbb{R}^d$ ,  $H_{uv}$  denotes the 2-dimensional linear subspace spanned by these vectors. Let  $\mathbf{u}^0$  and  $\mathbf{u}^1$  be projections of vectors  $\mathbf{u}^0$  and  $\mathbf{u}^1$  (generated at Step 1 in Algorithm RR) onto  $H_{uv}$ . Then the randomness of vectors  $\mathbf{u}^0$  and  $\mathbf{u}^1$  implies that the normalized vectors of  $\mathbf{u}^0$  and  $\mathbf{u}^1$ , defined by  $\mathbf{u}^0/\|\mathbf{u}^0\|$  and  $\mathbf{u}^1/\|\mathbf{u}^1\|$ , are uniformly distributed on a unit circle in the plane  $H_{uv}$ . Thus, we have that  $\forall (u, v) \in V^2, \forall i \in \{0, 1\}$ , a hyperplane  $H = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{u}^i \top \mathbf{x} = 0\}$  does not separate two points  $\tilde{\mathbf{x}}(u), \tilde{\mathbf{x}}(v) \in \mathbb{R}^d$  with the probability

$$\Pr[\exists s \in \{+, -\}, \{u, v\} \subseteq V_i^s] = 1 - \frac{\tilde{\theta}_{uv}}{\pi}.$$

The independency of  $\{\mathbf{u}^0, \mathbf{u}^1\}$  induces that for any mutually different pair of polygons  $u, v \in V$ ;

$$\begin{aligned}
 \Pr[\{u, v\} \subseteq E_0] &= \left( 1 - \frac{\tilde{\theta}_{uv}}{\pi} \right)^2, \\
 \Pr[\{u, v\} \subseteq E_1] &= 2 \left( \frac{\tilde{\theta}_{uv}}{\pi} \right) \left( 1 - \frac{\tilde{\theta}_{uv}}{\pi} \right), \\
 \Pr[\{u, v\} \subseteq E_2] &= \left( \frac{\tilde{\theta}_{uv}}{\pi} \right)^2.
 \end{aligned}$$

Now we have the following lemma, which plays an important role in estimating an approximation ratio.

**Lemma 1.** *If an angle  $\theta$  satisfies  $0 \leq \forall \theta \leq \frac{2\pi}{3}$ , then the following inequalities:*

$$\begin{aligned} \left(\frac{\theta}{\pi}\right)^2 + \left(\frac{3}{4}\right) 2\left(\frac{\theta}{\pi}\right)\left(1 - \frac{\theta}{\pi}\right) &\geq \left(\frac{7}{9}\right) \left(\frac{-2 \cos \theta}{3} + \frac{2}{3}\right), \\ \left(1 - \frac{\theta}{\pi}\right)^2 + \left(\frac{1}{4}\right) 2\left(\frac{\theta}{\pi}\right)\left(1 - \frac{\theta}{\pi}\right) &\geq \left(\frac{7}{9}\right) \left(\frac{2 \cos \theta}{3} + \frac{1}{3}\right), \end{aligned}$$

hold.

**Proof.** The above inequalities follow by simple calculus (see Figure 4). □

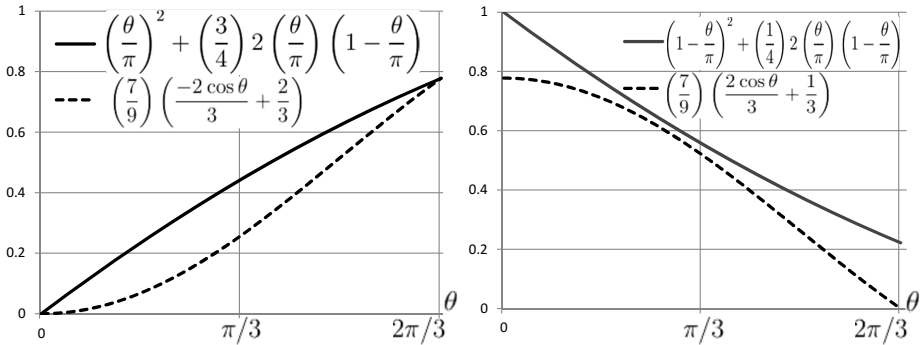


Fig. 4. Approximation ratio

The above Lemma directly implies the following lower bound of  $E[\tilde{Z}]$ :

$$\begin{aligned} E[\tilde{Z}] &\geq \alpha_1 \sum_{\{u,v\} \in E_C} \left( \left(\frac{\tilde{\theta}_{uv}}{\pi}\right)^2 + \left(\frac{3}{4}\right) 2\left(\frac{\tilde{\theta}_{uv}}{\pi}\right)\left(1 - \frac{\tilde{\theta}_{uv}}{\pi}\right) \right) \\ &\quad + \alpha_2 \sum_{\{u,v\} \in E_S} \left( \left(1 - \frac{\tilde{\theta}_{uv}}{\pi}\right)^2 + \left(\frac{1}{4}\right) 2\left(\frac{\tilde{\theta}_{uv}}{\pi}\right)\left(1 - \frac{\tilde{\theta}_{uv}}{\pi}\right) \right) \\ &\geq \alpha_1 \sum_{\{u,v\} \in E_C} \left(\frac{7}{9}\right) \left(\frac{-2 \cos \tilde{\theta}_{uv}}{3} + \frac{2}{3}\right) + \alpha_2 \sum_{\{u,v\} \in E_S} \left(\frac{7}{9}\right) \left(\frac{2 \cos \tilde{\theta}_{uv}}{3} + \frac{1}{3}\right) \\ &= \left(\frac{7}{9}\right) \left( \alpha_1 \sum_{\{u,v\} \in E_C} \left(\frac{-2\tilde{X}_{uv}}{3} + \frac{2}{3}\right) + \alpha_2 \sum_{\{u,v\} \in E_S} \left(\frac{2\tilde{X}_{uv}}{3} + \frac{1}{3}\right) \right) \\ &= \left(\frac{7}{9}\right) z(\text{SDP}) \geq \left(\frac{7}{9}\right) z(\text{P1}), \end{aligned}$$

where  $z(\text{SDP})$  and  $z(\text{P1})$  denote the optimal values of SDP and P1, respectively. From the above discussion, we have the following theorem.

**Theorem 1.** *For any problem instance of P1, the expectation of the objective value corresponding to a 3-coloring obtained by Algorithm RR is greater than or equal to  $(7/9)$  times the optimal value of the original problem P1.*

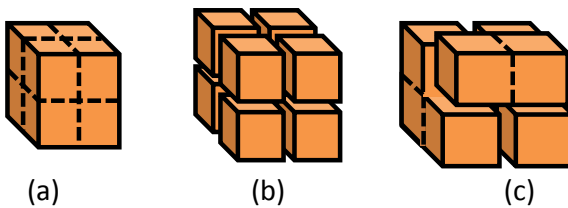
## 7 Discussions

In this paper, we proposed an approximation algorithm for TPL layout decomposition, which opens a new theoretical perspective of methods for TPL with guaranteed accuracy.

We formulate a problem of TPL layout decomposition which maximizes objective value representing decomposition quality. Our maximization problem is essentially equivalent to an ordinary problem of minimizing weighted sum of the number of conflicts and number of used stitch candidates [2, 5, 6, 14, 15].

Comparing to existing methods, our algorithm has a theoretical precision of an output, i.e., the expectation of the corresponding objective function value is greater than or equal to  $(7/9)$  times the optimal value of original problem even in the worst case problem instance. Our result is the first approximation algorithm with a constant approximation ratio for TPL layout decomposition.

Algorithm RR applies the hyperplane separation technique twice and obtain a partition of vertices consisting of four vertex subsets (see Figure 3 (a)). If we apply the hyperplane separation technique three times, we obtain a partition of vertices consisting of eight vertex subsets (see (a) and (b) of Figure 5). Figure 5 (c) indicates a rounding technique for generating vertex 3-coloring, which has 12 variations obtained by rotating Figure 5. If we execute both Algorithm RR and the rounding method indicated by Figure 5, then a better solution gives a vertex 3-coloring, whose approximation ratio is greater than 0.78855. (Here we omit the detail due to limitations of space.)



**Fig. 5.** Rounding technique for a partition obtained by three hyperplanes

We can extend the existing method proposed by Yu et al. in [14] to a method with guaranteed accuracy by simply adding our randomized rounding procedure and selecting a better solution (from a pair of solutions obtained by the existing method and our randomized rounding procedure). Here we note that we need not to solve SDP twice. Even in the case that the existing method finds a better

solution, the above synthesis gives a certificate of accuracy to output. Since our randomized rounding procedure only requires positive semi-definiteness of a solution, one can apply our procedure to an optimal solution of existing positive semidefinite relaxation problems for TPL layout decomposition (e.g., [12]), even if its objective function includes different terms.

## References

1. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. In: Proc. IEEE Symposium on Foundations of Computer Science (FOCS), pp. 238–250 (2002)
2. Fang, S.-Y., Chang, Y.-W., Chen, W.-Y.: A novel layout decomposition algorithm for triple patterning lithography. In: Proc. ACM/IEEE Design Automation Conference (DAC), pp. 1181–1186 (2012)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. Freeman and Co., New York (1979)
4. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* **42**, 1115–1145 (1995)
5. Kuang, J., Young, E.: An efficient layout decomposition approach for triple patterning lithography. In: Proc. ACM/IEEE Design Automation Conference (DAC), pp. 1–6 (2013)
6. Lin, Y.-H., Yu, B., Pan, D.Z., Li, Y.-L.: TRIAD: a triple patterning lithography aware detailed router. In: Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 123–129 (2012)
7. Ma, Q., Zhang, H., Wong, M.D.F.: Triple patterning aware routing and its comparison with double patterning aware routing in 14 nm technology. In: Proc. ACM/IEEE Design Automation Conference (DAC), pp. 591–596 (2012)
8. Swamy, C.: Correlation clustering: maximizing agreements via semidefinite programming. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 526–527 (2004)
9. Tian, H., Du, Y., Zhang, H., Xiao, Z., Wong, M.D.F.: Constrained pattern assignment for standard cell based triple patterning lithography. In: Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 178–185 (2013)
10. Tian, H., Zhang, H., Ma, Q., Xiao, Z., Wong, M.D.F.: A polynomial time triple patterning algorithm for cell based row-structure layout. In: Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 57–64 (2012)
11. Yu, B., Gao, J.-R., Pan, D.Z.: Triple patterning lithography (TPL) layout decomposition using end-cutting. In: Proc. SPIE, vol. 8684, p. 86840G (2013)
12. Yu, B., Lin, Y.-H., Luk-Pat, G., Ding, D., Lucas, K., Pan, D.Z.: A high-performance triple patterning layout decomposer with balanced density. In: Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 163–169 (2013)
13. Yu, B., Xu, X., Gao, J.-R., Pan, D.Z.: Methodology for standard cell compliance and detailed placement for triple patterning lithography. In: Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 349–356 (2013)
14. Yu, B., Yuan, K., Zhang, B., Ding, D., Pan, D.Z.: Layout decomposition for triple patterning lithography. In: Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 1–8 (2011)
15. Zhang, Y., Luk, W.-S., Zhou, H., Yan, C., Zeng, X.: Layout decomposition with pairwise coloring for multiple patterning lithography. In: Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 170–177 (2013)

# An FPTAS for the Volume Computation of 0-1 Knapsack Polytopes Based on Approximate Convolution Integral

Ei Ando<sup>1</sup>(✉) and Shuji Kijima<sup>2</sup>

<sup>1</sup> Sojo University, 4-22-1, Ikeda, Nishi-Ku, Kumamoto 860-0082, Japan  
ando-ei@cis.sojo-u.ac.jp

<sup>2</sup> Kyushu University, 744, Motoooka, Nishi-Ku, Fukuoka 819-0395, Japan  
kijima@inf.kyushu-u.ac.jp

**Abstract.** Computing high dimensional volumes is a hard problem, even for approximation. It is known that no polynomial-time *deterministic* algorithm can approximate with ratio  $1.999^n$  the volumes of convex bodies in the  $n$  dimension as given by membership oracles. Several *randomized* approximation techniques for #P-hard problems has been developed in the three decades, while some deterministic approximation algorithms are recently developed only for a few #P-hard problems. For instance, Stefankovic, Vempala and Vigoda (2012) gave an FPTAS for counting 0-1 knapsack solutions (i.e., integer points in a 0-1 knapsack polytope) based on an ingenious dynamic programming. Motivated by a new technique for designing FPTAS for #P-hard problems, this paper is concerned with the *volume* computation of 0-1 *knapsack polytopes*: it is given by  $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} \leq b, 0 \leq x_i \leq 1 (i = 1, \dots, n)\}$  with a positive integer vector  $\mathbf{a}$  and a positive integer  $b$  as an input, the volume computation of which is known to be #P-hard. Li and Shi (2014) gave an FPTAS for the problem by modifying the dynamic programming for counting solutions. This paper presents a new technique based on *approximate convolution integral* for a *deterministic* approximation of volume computations, and provides an FPTAS for the volume computation of 0-1 knapsack polytopes.

**Keywords:** Approximate convolution integral · Volume computation · #P-hard · Knapsack polytope

## 1 Introduction

Computing a volume in  $n$ -dimensional space is a hard problem, even for approximation. Lovász [16] showed that no polynomial-time *deterministic* algorithm can

---

This work is partly supported by Grant-in-Aid for Scientific Research on Innovative Areas MEXT Japan “Exploring the Limits of Computation (ELC)” (No. 24106008, 24106005).

approximate with ratio  $1.999^n$  the volumes of convex bodies in the  $n$  dimensional space as given by membership oracles (see also [2, 7]).

Several *randomized* approximation techniques for #P-hard problems has been developed, such as the Markov chain Monte Carlo method. For the volume computation of general convex bodies given by a membership oracle in the  $n$  dimensional space, Dyer, Frieze and Kannan [6] gave the first *fully polynomial-time randomized approximation scheme (FPRAS)*, giving a rapidly mixing Markov chain. In fact, the running time of the FPRAS is  $O^*(n^{23})$  where  $O^*$  ignores  $\text{poly}(\log n)$  and  $1/\epsilon$  terms. Subsequently, several techniques have been developed for the volume computation, and Lovász and Vempala [17] finally gave an improved algorithm of  $O^*(n^4)$ -time.

In contrast, it is a major challenge to design *deterministic* approximation algorithms for #P-hard problems, and not many results seem to be known. A remarkable progress is the *correlation decay* argument due to Weitz [21]; he designed a *fully polynomial time approximation scheme (FPTAS)*, for counting independent sets in graphs of maximum degree  $\Delta \geq 5$ . A similar technique is independently presented by Bandyopadhyay and Gamarnik [1], and there are several recent developments on the technique, e.g., [3, 8, 12, 13, 15]. For counting 0-1 knapsack solutions, Gopalan, Klivans and Meka [9], and Stefankovic, Vempala and Vigoda [19] gave deterministic approximation algorithms based on the dynamic programming (see also [10]), in a similar way to a simple random sampling algorithm by Dyer [4]. Modifying the dynamic programming, Li and Shi [14] recently gave an FPTAS for computing a distribution function of sum of random variables, including the volume computation of 0-1 knapsack polytopes.

Motivated by a new technique of designing FPTASs for #P-hard problems, this paper is concerned with the *volume* computation of 0-1 knapsack polytopes: Given a positive integer<sup>1</sup> vector  $\mathbf{a}^\top = (a_1, \dots, a_n) \in \mathbb{Z}_{>0}^n$  and a positive integer  $b \in \mathbb{Z}_{>0}$ , the 0-1 *knapsack polytope* is given by

$$K(b) \stackrel{\text{def}}{=} \{ \mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} \leq b, 0 \leq x_i \leq 1 \ (i = 1, \dots, n) \}.$$

For the simplicity of description, we in the following assume that  $a_1 \leq \dots \leq a_n$ , without loss of generality. Computing the *volume* of  $K(b)$  is known to be #P-hard (see e.g., [5]). Remark that counting solutions corresponds to counting the integer points in  $K(b)$ , and it is different from the volume computation, but closely related.

This paper presents a new technique for designing a *deterministic* approximation algorithm for volume computations, and provides an FPTAS for the volume computation of 0-1 knapsack polytopes, i.e., our main result is the following:

**Theorem 1.1.** *For any  $\epsilon > 0$ , there exists an  $O(\tau n^4/\epsilon)$ -time deterministic algorithm to approximate  $\text{Vol}(K(b))$  with approximation ratio  $1 + \epsilon$ , where  $\tau$  denotes the complexity of a numerical computation, which is bounded by  $\tau = O(n \log(\max\{a_n, b\}))$  in the algorithm.*

<sup>1</sup> For the simplicity of arguments, this paper assumes integer, while our technique is also applied to real valued inputs. See e.g., [11, 20] for a treatment of real values.



The idea of our algorithm is based on the classical *convolution integral*, while there seems no known technique (explicitly) using an approximate convolution integral to design an FPTAS for #P-hard problems. In fact, our algorithm repeats a recursive approximate convolution with  $O(n^3/\epsilon)$  iterations for  $n$  times, meaning that the number of iteration is independent of the values  $a_i$  and  $b$ .

In comparison with Li and Shi [14], the running time of their algorithm is  $O((n^3/\epsilon^2) \log(1/\text{Vol}(K(b))) \log b)$  where notice that  $\text{Vol}(K(b)) \leq 1$  holds by the definition of  $K(b)$ . Their algorithm is based on the dynamic programming by [19], and then the number of *iterations* is  $\Omega(1/\epsilon)$  and  $\Omega(\log(\text{Vol}(K(b))))$ , i.e., depending on  $\Omega(\log a_i)$  or  $\Omega(\log b)$ , in their algorithm. In contrast, our technique is completely different, and the algorithm requires  $O(n^4/\epsilon)$  iterations; the algorithm is somehow combinatorial (or “strongly polynomial,” in a sense) rather than arithmetic.

This paper is organized as follows. In Section 2, we describe the volume computation in terms of the recursive convolution integral, to explain the idea of our algorithm. In Section 3, we explain our algorithm, and give an analysis under the assumption that  $a_i \leq b$  ( $i = 1, \dots, n$ ). In Section 4, we briefly explain how to get rid of the assumption. In Section 5, we conclude the paper.

## 2 Distribution Function of Uniform Sum

Let  $\mathbf{Y} = (Y_1, \dots, Y_n)$  be a uniform random variable over  $[0, 1]^n$ . Then, it is not difficult to see that  $[\mathbf{a}^\top \mathbf{Y} \leq b]$  if and only if  $[\mathbf{Y} \in K(b)]$ , meaning that

$$\Pr[\mathbf{a}^\top \mathbf{Y} \leq b] = \Pr[\mathbf{Y} \in K(b)] = \frac{\text{Vol}(K(b))}{\text{Vol}([0, 1]^n)} = \text{Vol}(K(b))$$

hold. For the convenience of the later arguments, let  $X_i = a_i Y_i$  for each  $i \in \{1, \dots, n\}$ . Then, we observe that

$$\Pr[X_1 + \dots + X_n \leq b] = \Pr[\mathbf{Y} \in K(b)]$$

holds, meaning that  $\Pr[X_1 + \dots + X_n \leq b] = \text{Vol}(K(b))$ . Clearly, each  $X_i$  is uniformly distributed over  $[0, a_i]$ , and  $X_1, \dots, X_n$  are mutually independent. Let  $f_i(x)$  and  $F_i(x) = \Pr[X_i \leq x]$  respectively denote the probability density function and the probability distribution function of  $X_i$ , i.e.,

$$f_i(x) = \begin{cases} \frac{1}{a_i} & 0 \leq x \leq a_i, \\ 0 & \text{otherwise,} \end{cases} \quad F_i(x) = \begin{cases} 0 & x \leq 0, \\ \frac{x}{a_i} & 0 \leq x \leq a_i, \\ 1 & x \geq a_i. \end{cases}$$

Now, we define  $\Phi_0(x) = 0$  for  $x \leq 0$ , while  $\Phi_0(x) = 1$  for  $x > 0$ . Inductively, for  $x \in \mathbb{R}$  and  $i = 1, 2, \dots, n$ , we define

$$\Phi_i(x) \stackrel{\text{def}}{=} \int_{-\infty}^{+\infty} \Phi_{i-1}(s) f_i(x - s) ds.$$

**Proposition 2.1.** *For each  $i \in \{1, 2, \dots, n\}$  and for any  $x \in \mathbb{R}$ ,*

$$\Phi_i(x) = \Pr [X_1 + \dots + X_i \leq x]$$

*Proof.* To begin with, consider the case in which  $i = 1$ . By the definition of  $\Phi_0$ ,

$$\Phi_1(x) = \int_{-\infty}^{+\infty} \Phi_0(s) f_1(x - s) ds = \int_0^{+\infty} f_1(x - s) ds = \int_{-\infty}^x f_1(t) dt = \Pr[X_1 \leq x]$$

and we obtain the claim in the case.

Now, inductively assuming that  $\Phi_{i-1}(x) = \Pr [X_1 + \dots + X_{i-1} \leq x]$  holds, we prove the case of  $i$  ( $i = 1, 2, \dots, n$ ).

$$\begin{aligned} \Pr [X_1 + \dots + X_i \leq x] &= \int_{-\infty}^{+\infty} \Pr [X_1 + \dots + X_{i-1} \leq x - s] f_i(s) ds \\ &= \int_{-\infty}^{+\infty} \Phi_{i-1}(x - s) f_i(s) ds = \int_{-\infty}^{+\infty} \Phi_{i-1}(s) f_i(x - s) ds = \Phi_i(x). \end{aligned}$$

We obtain the claim. □

For convenience, let  $\Phi(b)$  denote  $\Phi_n(b)$ .

**Corollary 2.2.**  $\Phi(b) = \text{Vol}(K(b))$ .

### 3 Approximation Algorithm

For simplicity, we assume that  $a_n \leq b$  in this section. The assumption will be removed in Section 4.

#### 3.1 The Idea

To begin with, this subsection explains the idea of our algorithm, that is a recursive computation of approximate convolutions  $G_i(x)$ , in an analogy with  $\Phi_i(x)$ . Let  $G_0(x) = \Phi_0(x)$ , i.e.,  $G_0(x) = 0$  for  $x \leq 0$  while  $G_1(x) = 1$  for  $x > 0$ . Inductively assuming  $G_{i-1}(x)$  ( $i = 1, \dots, n$ ), we define

$$\bar{G}_i(x) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} G_{i-1}(s) f_i(x - s) ds$$

for  $x \in \mathbb{R}$ , as an intermediate of  $G_i(x)$  for convenience. Let  $r_i \stackrel{\text{def}}{=} \sum_{i'=1}^i a_{i'}$  (i.e.,  $r_i = \sup\{r \in \mathbb{R} \mid \bar{G}_i(r) < 1\}$  holds, in fact), and let  $M \in \mathbb{Z}_{>0}$  be a discretization parameter of the algorithm. Then, let  $G_i(x)$  be a staircase approximation of  $\bar{G}_i(x)$ , which is given by

$$G_i(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \bar{G}_i\left(\frac{j}{M}r_i\right) & \text{if } \frac{j-1}{M}r_i < x \leq \frac{j}{M}r_i \quad (j = 1, \dots, M), \\ 1 & \text{if } x > r_i. \end{cases}$$

Since  $G_i(x)$  is a staircase function, our algorithm only requires the values of  $G_i(\frac{j}{M}r_i) = \overline{G}_i(\frac{j}{M}r_i)$  for  $j = 1, \dots, M$ , instead of all  $x \in \mathbb{R}$ .

Here, we briefly discuss the computation of  $G_i(\frac{j}{M}r_i)$ , that is  $\overline{G}_i(\frac{j}{M}r_i)$ , for  $j = 1, \dots, M$  from  $G_{i-1}(x)$ . In the concerning case, that is  $0 < x \leq r_i$ , we see that

$$\begin{aligned} \overline{G}_i(x) &= \int_{-\infty}^{\infty} G_{i-1}(s)f_i(x-s)ds \\ &= \int_0^x G_{i-1}(s)f_i(x-s)ds \quad (\text{since } G_{i-1}(s) = 0 \text{ for } s \leq 0) \\ &= \int_{\max\{0, x-a_i\}}^x G_{i-1}(s) \cdot \frac{1}{a_i} ds \quad (\text{consider } s \text{ s.t. } f_i(x-s) \neq 0) \end{aligned}$$

hold. Thus,

$$G_i\left(\frac{j}{M}r_i\right) = \overline{G}_i\left(\frac{j}{M}r_i\right) = \frac{1}{a_i} \int_{\max\{0, \frac{j}{M}r_i - a_i\}}^{\frac{j}{M}r_i} G_{i-1}(s) ds$$

holds. For each  $j \in \{1, \dots, M\}$ , the following four cases are considered:

- Case 1:**  $j$  satisfies  $\frac{j}{M}r_i < a_i$  and  $\frac{j}{M}r_i \leq r_{i-1}$ ,
- Case 2:**  $j$  satisfies  $\frac{j}{M}r_i < a_i$  and  $\frac{j}{M}r_i > r_{i-1}$ ,
- Case 3:**  $j$  satisfies  $\frac{j}{M}r_i \geq a_i$  and  $\frac{j}{M}r_i \leq r_{i-1}$ ,
- Case 4:**  $j$  satisfies  $\frac{j}{M}r_i \geq a_i$  and  $\frac{j}{M}r_i > r_{i-1}$ .

In the Case 1, for instance, let  $j^* = \max\{j' \in \{1, \dots, M\} \mid \frac{j'}{M}r_{i-1} < \frac{j}{M}r_i\}$  (i.e.,  $j^* = \lceil j \frac{r_i}{r_{i-1}} \rceil$ ), and then we obtain that

$$\begin{aligned} G_i\left(\frac{j}{M}r_i\right) &= \frac{1}{a_i} \int_0^{\frac{j}{M}r_i} G_{i-1}(s) ds = \frac{1}{a_i} \int_0^{\frac{j^*}{M}r_{i-1}} G_{i-1}(s) ds + \frac{1}{a_i} \int_{\frac{j^*}{M}r_{i-1}}^{\frac{j}{M}r_i} G_{i-1}(s) ds \\ &= \frac{1}{a_i} \sum_{j'=1}^{j^*} \int_{\frac{j'-1}{M}r_{i-1}}^{\frac{j'}{M}r_{i-1}} G_{i-1}(s) ds + \frac{1}{a_i} \cdot \left(\frac{j}{M}r_i - \frac{j^*}{M}r_{i-1}\right) \cdot G_{i-1}\left(\frac{j}{M}r_i\right) \\ &= \frac{1}{a_i} \sum_{j'=1}^{j^*} \frac{r_{i-1}}{M} \cdot G_{i-1}\left(\frac{j'}{M}r_{i-1}\right) + \frac{1}{a_i} \cdot \left(\frac{j}{M}r_i - \frac{j^*}{M}r_{i-1}\right) \cdot G_{i-1}\left(\frac{j}{M}r_i\right) \end{aligned}$$

where we remark that  $\int_{\frac{j^*}{M}r_{i-1}}^{\frac{j}{M}r_i} G_{i-1}(s) ds = \frac{r_{i-1}}{M} \cdot G_{i-1}\left(\frac{j}{M}r_{i-1}\right)$  since  $G_{i-1}(x)$  is a staircase function. Thus, it is not difficult to see that we can compute  $G_i(\frac{j}{M}r_i)$  in  $O(M\tau)$  time where  $\tau$  denotes the time complexity of a numerical computation.

Similarly, we can compute  $G_i(\frac{j}{M}r_i)$  in  $O(M\tau)$  time, in other Cases 2–4.

### 3.2 Algorithm and Analysis

Based on the arguments in Section 3.1, our algorithm is described as follows.

**Algorithm 1**

- Input:  $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{Z}_{>0}^n$  and  $b \in \mathbb{Z}_{>0}$ .
1. Let  $G_0(x) := 0$  for  $x \leq 0$ , and let  $G_0(x) := 1$  for  $x > 0$ ;
  2. For  $i = 1, \dots, n$
  3.     For  $j = 1, \dots, M$
  4.         Compute  $G_i(\frac{j}{M}r_i)$ ;
  5. Output  $G_n(b)$ .

For convenience, let  $G(b)$  denote  $G_n(b)$ , in the following. Firstly, we discuss the time complexity of Algorithm 1.

**Proposition 3.1.** *The running time of Algorithm 1 is  $O(nM^2\tau)$ .*

*Proof.* As stated in Section 3.1,  $G_i(\frac{j}{M}r_i)$  is computed at line 4 in  $O(M\tau)$  time. Now, the claim is easy. □

In fact, we can improve the time complexity to  $O(nM\tau)$ , improving the time complexity of lines 3 and 4 into  $O(M\tau)$  by inductively computing  $G_i(\frac{j}{M}r_i)$  using the result  $G_i(\frac{j-1}{M}r_i)$ . It is somehow complicated, but straightforward.

**Lemma 3.2.** *The running time of Algorithm 1 is  $O(nM\tau)$ .*

Next, we establish the following approximation ratio of Algorithm 1, which we will prove in the next subsection.

**Lemma 3.3.** *For any  $\epsilon > 0$ , set  $M \geq \frac{n^2(n+1)}{2}\epsilon^{-1}$ , then we have*

$$\Phi(b) \leq G(b) \leq (1 + \epsilon)\Phi(b).$$

Now, Theorem 1.1 is immediate from Lemmas 3.2 and 3.3.

**3.3 Proof of Lemma 3.3, for Approximation Ratio**

As a preliminary, we observe the following facts from the definitions of  $\bar{G}_i(x)$  and  $G_i(x)$ .

**Observation 3.4.** *For any  $i \in \{1, 2, \dots, n\}$ ,  $\Phi_i(x)$ ,  $\bar{G}_i(x)$  and  $G_i(x)$  are monotone nondecreasing (with respect to  $x$ ), respectively.*

**Observation 3.5.** *For each  $i \in \{1, 2, \dots, n\}$ ,  $\bar{G}_i(x) \leq G_i(x) \leq \bar{G}_i(x + \frac{1}{M}r_i)$  holds for any  $x \in \mathbb{R}$ .*

**Proposition 3.6.** *For each  $i \in \{1, 2, \dots, n\}$ ,  $\Phi_i(x) \leq \bar{G}_i(x)$  for any  $x \in \mathbb{R}$ .*

*Proof.* By the definition,  $\Phi_0(x) = \bar{G}_0(x)$  for any  $x \in \mathbb{R}$ . Inductively assuming that  $\Phi_{i-1}(x) \leq \bar{G}_{i-1}(x)$  holds for any  $x \in \mathbb{R}$ , we have that

$$\begin{aligned} \Phi_i(x) &= \int_{-\infty}^{\infty} \Phi_{i-1}(s)f_i(x-s)ds \leq \int_{-\infty}^{\infty} \bar{G}_{i-1}(s)f_i(x-s)ds \quad (\text{Induction hypo.}) \\ &\leq \int_{-\infty}^{\infty} G_{i-1}(s)f_i(x-s)ds = \bar{G}_i(x) \quad (\text{by Obs 3.5}) \end{aligned}$$

for any  $x \in \mathbb{R}$ . □

Now, we establish the following Lemma 3.7, which claims a “horizontal” approximation ratio, in spite of Lemma 3.3 claiming a “vertical” bound. For convenience, let  $\ell_i \stackrel{\text{def}}{=} \sum_{j=1}^i r_j/M$ , i.e.,

$$\ell_i = \sum_{j=1}^i \frac{\sum_{j'=1}^j a_{j'}}{M} = \frac{\sum_{j=1}^i \sum_{j'=1}^j a_{j'}}{M} = \frac{\sum_{j=1}^i j a_j}{M} \leq \frac{i(i+1)}{2} \cdot \frac{a_i}{M} \tag{1}$$

where the last inequality follows from the assumption that  $a_1 \leq \dots \leq a_n$ .

**Lemma 3.7.** *For each  $i \in \{1, \dots, n\}$  and any  $x \in \mathbb{R}$ , we have*

$$\Phi_i(x) \leq G_i(x) \leq \Phi_i(x + \ell_i).$$

*Proof.* The former inequality is immediate from Proposition 3.6 and Observation 3.5. Now, we are concerned with the latter inequality. Observation 3.5 implies that

$$G_1(x) \leq \bar{G}_1\left(x + \frac{r_1}{M}\right) = \Phi_1\left(x + \frac{r_1}{M}\right),$$

and obtain the claim for  $i = 1$ . Inductively assuming that  $G_{i-1}(x) \leq \Phi_{i-1}(x + \ell_{i-1})$  for any  $x \in \mathbb{R}$ , we show that  $G_i(x) \leq \Phi_i(x + \ell_i)$ . Notice that

$$\begin{aligned} \bar{G}_i(x) &= \int_{-\infty}^{\infty} G_{i-1}(s) f_i(x-s) ds \leq \int_{-\infty}^{\infty} \Phi_{i-1}(s + \ell_{i-1}) f_i(x-s) ds \quad (\text{induction hypo.}) \\ &= \int_{-\infty}^{\infty} \Phi_{i-1}(t) f_i(x + \ell_{i-1} - t) dt \quad (t := s + \ell_{i-1}) \\ &= \Phi_i(x + \ell_{i-1}) \end{aligned} \tag{2}$$

hold for any  $x \in \mathbb{R}$ . Using Observation 3.5, (2) implies that

$$G_i(x) \leq \bar{G}_i\left(x + \frac{r_i}{M}\right) \leq \Phi_i\left(x + \frac{r_i}{M} + \ell_{i-1}\right) = \Phi_i(x + \ell_i)$$

where the last equality follows from the definition of  $\ell_i$ . We obtain the claim.  $\square$

The following Lemma 3.8 will be established in Section 3.4.

**Lemma 3.8.** *Let  $\phi(x) \stackrel{\text{def}}{=} \frac{d}{dx} \Phi(x)$  (recall that  $\Phi(x) = \Phi_n(x)$ ). Then, for any  $x > 0$ , we have*

$$\frac{\phi(x)}{\Phi(x)} \leq \frac{n}{x}.$$

Now, we show Lemma 3.3, using Lemmas 3.7 and 3.8.

*Proof (of Lemma 3.3).* The first inequality  $\Phi(b) \leq G(b)$  is immediate from Lemma 3.7. Thus, we give an upper bound of  $\Phi(b + \ell_n)/\Phi(b)$ . By the mean value theorem, there exists  $c \in \mathbb{R}$  satisfying  $b \leq c \leq b + \ell_n$ , such that  $\Phi(b +$

$\ell_n = \Phi(b) + \phi(c)\ell_n$  holds. Let  $\beta$  be either one of  $b, c$  or  $b + \ell_n$  achieving that  $\phi(\beta) = \max\{\phi(b), \phi(c), \phi(b + \ell_n)\}$ . Then,

$$\begin{aligned} \frac{\Phi(b)}{\Phi(b + \ell_n)} &= \frac{\Phi(b + \ell_n) - \phi(c)\ell_n}{\Phi(b + \ell_n)} = 1 - \frac{\phi(c)\ell_n}{\Phi(b + \ell_n)} \\ &\geq 1 - \frac{\phi(\beta)\ell_n}{\Phi(b + \ell_n)} \geq 1 - \frac{\phi(\beta)\ell_n}{\Phi(\beta)} \end{aligned} \tag{3}$$

hold, where the last inequality follows that  $\Phi(x)$  is monotone nondecreasing with respect to  $x$  (Observation 3.4), and that  $b \leq \beta$ . Since Lemma 3.8 implies that  $\frac{\phi(\beta)}{\Phi(\beta)} \leq \frac{n}{\beta}$ , using (1) that is  $\ell_n \leq \frac{n(n+1)}{2} \cdot \frac{a_n}{M}$ ,

$$\begin{aligned} (3) &\geq 1 - \frac{n}{\beta} \cdot \frac{n(n+1)a_n}{2M} \geq 1 - \frac{n^2(n+1)}{2M} \quad (\text{since } a_n \leq b \leq \beta) \\ &\geq 1 - \epsilon \quad \left(\text{since } M \geq \frac{n^2(n+1)}{2}\epsilon^{-1}\right). \end{aligned}$$

We obtain the claim since we have  $1 + (\gamma + 1)\epsilon \geq 1/(1 - \epsilon)$  for any constant  $\gamma \geq \epsilon$ . For example, we have

$$1 + 2\epsilon \geq \frac{1}{1 - \epsilon} \geq \frac{\Phi(b + \ell_n)}{\Phi(b)},$$

for  $\epsilon \leq 1$ . □

### 3.4 Proof of Lemma 3.8

We use the following definition and lemmas in the proof of Theorem 1.1.

**Definition 3.9.** For given  $\mathbf{a} \in \mathbb{Z}_{>0}^n$  and  $b \in \mathbb{Z}_{>0}$ , let  $P(b) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} = b\}$ . For a bounded point set  $B \subset P(b)$ , the area  $\text{Area}(B)$  of  $B$  is given by

$$\text{Area}(B) = \int_{\mathbf{x} \in P(b)} \mu(B, \mathbf{x}) d\mathbf{x}, \tag{4}$$

where  $\mu(B, \mathbf{x})$  is a function satisfying  $\mu(B, \mathbf{x}) = 1$  if  $\mathbf{x} \in B$ , and  $\mu(B, \mathbf{x}) = 0$  otherwise.

**Lemma 3.10.** Consider, for a real number  $x \in \mathbb{R}$ , any bounded point set  $B \subseteq P(b)$  and a cone  $C \subseteq \mathbb{R}^n$  that is given by  $B$  and the points in between  $B$  and the origin. That is,  $C = \{\mathbf{x} \in \mathbb{R}^n \mid \exists c \in \mathbb{R}, \text{ s.t. } c\mathbf{x} \in B, 0 \leq c \leq 1\}$ . Then we have

$$\frac{\text{Area}(B)}{\text{Vol}(C)} = \frac{n}{b}.$$

Fig. 1 shows an example of  $B$  and  $C$  in the case of  $n = 3$ .

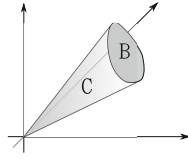


Fig. 1. An example of  $B$  and  $C$  in the case  $n = 3$

*Proof.* Consider a point set  $B'(t) = \{\mathbf{x} \in \mathbb{R}^n \mid (t/b)\mathbf{x} \in B\}$  on hyperplane  $P(t)$  and a cone  $C'(t) = \{\mathbf{x} \in \mathbb{R}^n \mid \exists c \in \mathbb{R}, \text{ s.t. } c\mathbf{x} \in B'(t), 0 \leq c \leq 1\}$  for  $t \in \mathbb{R}$ . Since  $C'(t)$  is obtained by scaling  $C'(1)$ , we have that

$$\text{Vol}(C'(t)) = t^n \text{Vol}(C'(1)).$$

Then, we have

$$\text{Area}(B'(t)) = \frac{d}{dt} \text{Vol}(C'(t)) = nt^{n-1} \text{Vol}(C'(1)),$$

Since  $B = B'(b)$  and  $C = C'(b)$ , the lemma is proved. □

*Proof (of Lemma 3.8).* Let  $B$  be a point set

$$B = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} = b, 0 \leq x_i \leq 1 \ (i = 1, \dots, n)\},$$

where  $\mathbf{e} = (1, \dots, 1)$  and  $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{Z}_{>0}^n$ . Then, let  $C$  be a cone given by

$$C = \{\mathbf{x} \in \mathbb{R}^n \mid \exists c \in \mathbb{R} \text{ s.t. } c\mathbf{x} \in B, 0 \leq c \leq 1\}.$$

By Lemma 3.10, we have that  $\text{Area}(B)/\text{Vol}(C) = n/b$ .

Since  $\text{Area}(B) = \phi(b)$  and  $\text{Vol}(C) \leq \Phi(b)$  ( $\because C \subseteq K(b)$ ), the lemma is proved. □

### 4 In Case of $b < a_n$

Here we consider how to deal with the cases where the input has parameters that are larger than  $b$ . Let  $a_1 \leq \dots \leq a_{n'} \leq b \leq a_{n'+1} \leq \dots \leq a_n$  and  $\mathbf{a}_{n',n} = (a_{n'+1}, \dots, a_n) \in \mathbb{R}^{n-n'}$ . Then, by using the algorithm 1, we first symbolically compute  $G_{n'}(x)$  for  $a_1, \dots, a_{n'}$  and  $b$ . After that, we compute the volume  $S_{n-n'}(x)$  of an  $(n - n')$ -dimensional simplex  $S$  that is given by  $S = \{\mathbf{x} \in \mathbb{R}^{n-n'} \mid \mathbf{a}_{n-n'}^\top \mathbf{x} \leq b, x_i \geq 0 \ (i = 1, \dots, n)\}$ . Then, we output the value obtained by the following convolution

$$G(b) = \int_{-\infty}^b G_{n'}(b - s) \frac{d}{dx} S_{n-n'}(s) ds. \tag{5}$$

Since we can exactly compute  $S_{n-n'}(x)$  as

$$S_{n-n'}(x) = \frac{x^{n-n'}}{(n-n')!} \prod_{i=n'+1}^n \frac{1}{a_i} \quad (6)$$

in time  $O((n-n')\tau)$ , we have the same order of computational time as in the previous section. When  $G_{n'}(x)$  is given within approximation ratio  $1 + \epsilon$ , it is easy to see that the approximation ratio of the above (5) is at most  $1 + \epsilon$ . This is verified by checking that  $\Phi_n(b + \ell_{n'}) \geq G(b) \geq \Phi_n(b)$ . Note that the horizontal difference between the upper bound and the lower bound is not  $\ell_n$  but  $\ell_{n'}$  because  $S_{n-n'}(x)$  is exact for  $x \leq b \leq a_{n'+1}$ .

## 5 Concluding Remarks

Motivated by a new technique of designing FPTAS for #P-hard problems, we have presented an FPTAS for computing the 0-1 knapsack volume, based on recursive approximate convolutions of  $O(n^4/\epsilon)$  iterations. Applications of the proposing technique to other #P-hard problems are future work.

## References

1. Bandyopadhyay, A., Gamarnik, D.: Counting without sampling: asymptotics of the log-partition function for certain statistical physics models. *Random Structures and Algorithms* **33**, 452–479 (2008)
2. Bárány, I., Füredi, Z.: Computing the volume is difficult. *Discrete Computational Geometry* **2**, 319–326 (1987)
3. Bayati, M., Gamarnik, D., Katz, D., Nair, C., Tetali, P.: Simple deterministic approximation algorithms for counting matchings. In: *Proc. of STOC 2007*, pp. 122–127 (2007)
4. Dyer, M.: Approximate counting by dynamic programming. In: *Proc. of STOC 2003*, pp. 693–699 (2003)
5. Dyer, M., Frieze, A.: On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing* **17**(5), 967–974 (1988)
6. Dyer, M., Frieze, A., Kannan, R.: A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the Association for Computing Machinery* **38**(1), 1–17 (1991)
7. Elekes, G.: A geometric inequality and the complexity of computing volume. *Discrete Computational Geometry* **1**, 289–292 (1986)
8. Gamarnik, D., Katz, D.: Correlation decay and deterministic FPTAS for counting list-colorings of a graph. In: *Proc. of SODA 2007*, pp. 1245–1254 (2007)
9. Gopalan, P., Klivans, A., Meka, R.: Polynomial-time approximation schemes for knapsack and related counting problems using branching programs, [arXiv:1008.3187v1](https://arxiv.org/abs/1008.3187v1) (2010)
10. Gopalan, P., Klivans, A., Meka, R., Štefankovič, D., Vempala, S., Vigoda, E.: An FPTAS for #knapsack and related counting problems. In: *Proc. of FOCS 2011*, pp. 817–826 (2011)
11. Ko, K.-I.: *Complexity Theory of Real Functions*. Birkhäuser, Boston (1991)



12. Li, L., Lu, P., Yin, Y.: Approximate counting via correlation decay in spin systems. In: Proc. of SODA 2012, pp. 922–940 (2012)
13. Li, L., Lu, P., Yin, Y.: Correlation decay up to uniqueness in spin systems. In: Proc. of SODA 2013, pp. 67–84 (2013)
14. Li, J., Shi, T.: A fully polynomial-time approximation scheme for approximating a sum of random variables. *Operations Research Letters* **42**, 197–202 (2014)
15. Lin, C., Liu, J., Lu, P.: A simple FPTAS for counting edge covers. In: Proc. of SODA 2014, pp. 341–348 (2014)
16. Lovász, L.: *An Algorithmic Theory of Numbers, Graphs and Convexity*. SIAM Society for industrial and applied mathematics, Philadelphia (1986)
17. Lovász, L., Vempala, S.: Simulated annealing in convex bodies and an  $O^*(n^4)$  volume algorithm. *Journal of Computer and System Sciences* **72**, 392–417 (2006)
18. Mitra, S.: On the probability distribution of the sum of uniformly distributed random variables. *SIAM Journal on Applied Mathematics* **20**(2), 195–198 (1971)
19. Štefankovič, D., Vempala, S., Vigoda, E.: A deterministic polynomial-time approximation scheme for counting knapsack solutions. *SIAM Journal on Computing* **41**(2), 356–366 (2012)
20. Weihrauch, K.: *Computable Analysis An Introduction*. Springer, Berlin (2000)
21. Weitz, D.: Counting independent sets up to the tree threshold. In: Proc. STOC 2006, pp. 140–149 (2006)

# **Graph Theory and Algorithms**

# Polynomial-Time Algorithm for Sliding Tokens on Trees

Erik D. Demaine<sup>1</sup>, Martin L. Demaine<sup>1</sup>, Eli Fox-Epstein<sup>2</sup>, Duc A. Hoang<sup>3</sup>,  
Takehiro Ito<sup>4</sup>(✉), Hiroataka Ono<sup>5</sup>, Yota Otachi<sup>3</sup>,  
Ryuhei Uehara<sup>3</sup>, and Takeshi Yamada<sup>3</sup>

<sup>1</sup> MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, USA  
{edemaine,mdemaine}@mit.edu

<sup>2</sup> Department of Computer Science, Brown University, Providence, USA  
ef@cs.brown.edu

<sup>3</sup> School of Information Science, JAIST, Nomi, Japan  
{hoanganhduc,otachi,uehara,tyama}@jaist.ac.jp

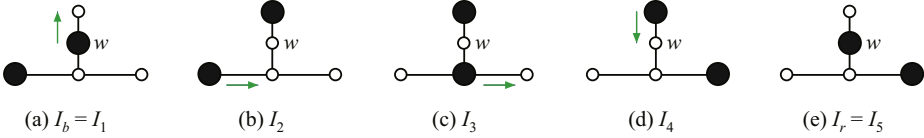
<sup>4</sup> Graduate School of Information Sciences, Tohoku University, Sendai, Japan  
takehiro@ecei.tohoku.ac.jp

<sup>5</sup> Faculty of Economics, Kyushu University, Fukuoka, Japan  
hirotaka@econ.kyushu-u.ac.jp

**Abstract.** Suppose that we are given two independent sets  $I_b$  and  $I_r$  of a graph such that  $|I_b| = |I_r|$ , and imagine that a token is placed on each vertex in  $I_b$ . Then, the SLIDING TOKEN problem is to determine whether there exists a sequence of independent sets which transforms  $I_b$  and  $I_r$  so that each independent set in the sequence results from the previous one by sliding exactly one token along an edge in the graph. This problem is known to be PSPACE-complete even for planar graphs, and also for bounded treewidth graphs. In this paper, we show that the problem is solvable for trees in quadratic time. Our proof is constructive: for a yes-instance, we can find an actual sequence of independent sets between  $I_b$  and  $I_r$  whose length (i.e., the number of token-slides) is quadratic. We note that there exists an infinite family of instances on paths for which any sequence requires quadratic length.

## 1 Introduction

Recently, *reconfiguration problems* attract the attention in the field of theoretical computer science. The problem arises when we wish to find a step-by-step transformation between two feasible solutions of a problem such that all intermediate results are also feasible and each step abides by a fixed reconfiguration rule (i.e., an adjacency relation defined on feasible solutions of the original problem). This kind of reconfiguration problem has been studied extensively for several well-known problems, including INDEPENDENT SET [4, 6, 9–12, 15, 17, 18, 20], SATISFIABILITY [8, 16], SET COVER, CLIQUE, MATCHING [11], VERTEX-COLORING [2, 5, 20], LIST  $L(2, 1)$ -LABELING [13], SHORTEST PATH [3, 14], and so on. (See also a recent survey [19].)



**Fig. 1.** A sequence  $\langle I_1, I_2, \dots, I_5 \rangle$  of independent sets of the same graph, where the vertices in independent sets are depicted by large black circles (tokens)

**1.1 SLIDING TOKEN**

The SLIDING TOKEN problem was introduced by Hearn and Demaine [9] as a one-player game, which can be seen as a reconfiguration problem for INDEPENDENT SET. Recall that an *independent set* of a graph  $G$  is a vertex-subset of  $G$  in which no two vertices are adjacent. (Figure 1 depicts five different independent sets in the same graph.) Suppose that we are given two independent sets  $I_b$  and  $I_r$  of a graph  $G = (V, E)$  such that  $|I_b| = |I_r|$ , and imagine that a token (coin) is placed on each vertex in  $I_b$ . Then, the SLIDING TOKEN problem is to determine whether there exists a sequence  $\langle I_1, I_2, \dots, I_\ell \rangle$  of independent sets of  $G$  such that

- (a)  $I_1 = I_b$ ,  $I_\ell = I_r$ , and  $|I_i| = |I_b| = |I_r|$  for all  $i$ ,  $1 \leq i \leq \ell$ ; and
- (b) for each  $i$ ,  $2 \leq i \leq \ell$ , there is an edge  $\{u, v\}$  in  $G$  such that  $I_{i-1} \setminus I_i = \{u\}$  and  $I_i \setminus I_{i-1} = \{v\}$ , that is,  $I_i$  can be obtained from  $I_{i-1}$  by sliding exactly one token on a vertex  $u \in I_{i-1}$  to its adjacent vertex  $v$  along  $\{u, v\} \in E$ .

Such a sequence is called a *reconfiguration sequence* between  $I_b$  and  $I_r$ . Figure 1 illustrates a reconfiguration sequence  $\langle I_1, I_2, \dots, I_5 \rangle$  of independent sets which transforms  $I_b = I_1$  into  $I_r = I_5$ . Hearn and Demaine proved that SLIDING TOKEN is PSPACE-complete for planar graphs, as an example of the application of their powerful tool, called the nondeterministic constraint logic model, which can be used to prove PSPACE-hardness of many puzzles and games [9], [10, Sec. 9.5].

**1.2 Related and Known Results**

As the (ordinary) INDEPENDENT SET problem is a key problem among thousands of NP-complete problems, SLIDING TOKEN plays a very important role since several PSPACE-hardness results have been proved using reductions from it. Indeed, SLIDING TOKEN is one of the most well-studied reconfiguration problems.

In addition, reconfiguration problems for INDEPENDENT SET (ISRECONF, for short) have been studied under different reconfiguration rules, as follows.

- *Token Sliding* (TS rule) [5,6,9,10,15,20]: This rule corresponds to the SLIDING TOKEN problem, that is, we can slide a single token only along an edge of a graph.
- *Token Jumping* (TJ rule) [6,12,15,20]: A single token can “jump” to any vertex (including non-adjacent one) if it results in an independent set.
- *Token Addition and Removal* (TAR rule) [4,11,15,17,18,20]: We can either add or remove a single token at a time if it results in an independent set of cardinality at least a given threshold. Therefore, under the TAR rule, independent sets in the sequence do not have the same cardinality.



**Fig. 2.** A yes-instance for ISRECONF under the TJ rule, which is a no-instance for the SLIDING TOKEN problem

We note that the existence of a desired sequence depends deeply on the reconfiguration rules. (See Fig. 2 for example.) However, ISRECONF is PSPACE-complete under any of the three reconfiguration rules for planar graphs [5, 9, 10], for perfect graphs [15], and for bounded bandwidth graphs [20]. The PSPACE-hardness implies that, unless  $NP = PSPACE$ , there exists an instance of SLIDING TOKEN which requires a super-polynomial number of token-slides even in a minimum-length reconfiguration sequence. In such a case, tokens should make “detours” to avoid violating independence. (For example, see the token placed on the vertex  $w$  in Fig. 1(a); it is moved twice even though  $w \in I_b \cap I_r$ .)

We here explain only the results which are strongly related to this paper, that is, SLIDING TOKEN on trees; see the references above for the other results.

**Results for TS rule (SLIDING TOKEN)**

Kamiński et al. [15] gave a linear-time algorithm to solve SLIDING TOKEN for cographs (also known as  $P_4$ -free graphs). They also showed that, for any yes-instance on cographs, two given independent sets  $I_b$  and  $I_r$  have a reconfiguration sequence such that no token makes detour.

Very recently, Bonsma et al. [6] proved that SLIDING TOKEN can be solved in polynomial time for claw-free graphs. Note that neither cographs nor claw-free graphs contain trees as a (proper) subclass. Thus, the complexity status for trees was open under the TS rule.

**Results for trees**

In contrast to the TS rule, it is known that ISRECONF can be solved in linear time under the TJ and TAR rules for even-hole-free graphs [15], which include trees. Indeed, the answer is always “yes” under the two rules when restricted to even-hole-free graphs (as long as two given independent sets have the same cardinality for the TJ rule.) Furthermore, tokens never make detours in even-hole-free graphs under the TJ and TAR rules.

On the other hand, under the TS rule, tokens are required to make detours even in trees. (See Fig. 1.) In addition, there are no-instances for trees under TS rule. (See Fig. 2.) These make the problem much more complicated, and we think they are the main reasons why SLIDING TOKEN for trees was open, despite the recent intensive algorithmic research on ISRECONF [4, 6, 12, 15, 18].

**1.3 Our Contribution**

In this paper, we show that the SLIDING TOKEN problem can be solved in time  $O(n^2)$  for any tree  $T$  with  $n$  vertices. Therefore, we can conclude that ISRECONF for trees is in P under any of the three reconfiguration rules.

We give a constructive proof: for a yes-instance, we can find an actual reconfiguration sequence between two given independent sets whose length is  $O(n^2)$ . We note that there exists an infinite family of instances on paths for which any reconfiguration sequence requires  $\Omega(n^2)$  length.

We note that, since the treewidth of any graph  $G$  can be bounded by the bandwidth of  $G$ , the result of [20] implies that SLIDING TOKEN is PSPACE-complete for bounded treewidth graphs. (See [1] for the definition of treewidth.) Thus, there exists an instance on bounded treewidth graphs which requires a super-polynomial number of token-slides even in a minimum-length reconfiguration sequence unless  $\text{NP} = \text{PSPACE}$ . Therefore, it is remarkable that any yes-instance on a tree, whose treewidth is one, has an  $O(n^2)$ -length reconfiguration sequence even though trees certainly require to make detours to transform.

## 1.4 Technical Overview

We here explain our main ideas; formal descriptions will be given later.

We say that a token on a vertex  $v$  is “rigid” under an independent set  $I$  of a tree  $T$  if it cannot be slid at all, that is,  $v \in I'$  holds for *any* independent set  $I'$  of  $T$  which is reconfigurable from  $I$ . (For example, in Fig. 2, every token in the two independent sets is rigid.) Our algorithm is based on the following two key points.

- (1) In Lemma 1, we will give a simple but non-trivial characterization of rigid tokens, based on which we can find all rigid tokens of two given independent sets  $I_b$  and  $I_r$  in time  $O(n^2)$ . Note that, if  $I_b$  and  $I_r$  have different placements of rigid tokens, then it is a no-instance (Lemma 4).
- (2) Otherwise, we obtain a forest by deleting the vertices with rigid tokens together with their neighbors (Lemma 5). We will prove in Lemma 6 that the answer is “yes” as long as each tree in the forest contains the same number of tokens in  $I_b$  and  $I_r$ .

Due to the page limitation, we omit some proofs from this extended abstract.

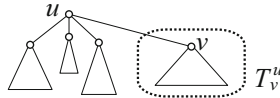
## 2 Preliminaries

In this section, we introduce some basic terms and notation.

### 2.1 Graph Notation

In the SLIDING TOKEN problem, we may assume without loss of generality that graphs are simple and connected. For a graph  $G$ , we sometimes denote by  $V(G)$  and  $E(G)$  the vertex set and edge set of  $G$ , respectively.

In a graph  $G$ , a vertex  $w$  is said to be a *neighbor* of a vertex  $v$  if  $\{v, w\} \in E(G)$ . For a vertex  $v$  in  $G$ , let  $N(G, v) = \{w \in V(G) \mid \{v, w\} \in E(G)\}$ . Let  $N[G, v] = N(G, v) \cup \{v\}$ . For a subset  $S \subseteq V(G)$ , we simply write  $N[G, S] = \bigcup_{v \in S} N[G, v]$ . For a subgraph  $G'$  of a graph  $G$ , we denote by  $G \setminus G'$  the subgraph of  $G$  induced by the vertices in  $V(G) \setminus V(G')$ .



**Fig. 3.** Subtree  $T_v^u$  in the whole tree  $T$

Let  $T$  be a tree. For two vertices  $v$  and  $w$  in  $T$ , the unique path between  $v$  and  $w$  is simply called the  $vw$ -path in  $T$ . We denote by  $\text{dist}(v, w)$  the number of edges in the  $vw$ -path in  $T$ . For two vertices  $u$  and  $v$  of a tree  $T$ , let  $T_v^u$  be the subtree of  $T$  obtained by regarding  $u$  as the root of  $T$  and then taking the subtree rooted at  $v$  which consists of  $v$  and all descendants of  $v$ . (See Fig. 3.) It should be noted that  $u$  is not contained in the subtree  $T_v^u$ .

**2.2 Definitions for SLIDING TOKEN**

Let  $I_i$  and  $I_j$  be two independent sets of a graph  $G$  such that  $|I_i| = |I_j|$ . If there exists exactly one edge  $\{u, v\}$  in  $G$  such that  $I_i \setminus I_j = \{u\}$  and  $I_j \setminus I_i = \{v\}$ , then we say that  $I_j$  can be obtained from  $I_i$  by *sliding* the token on  $u \in I_i$  to its adjacent vertex  $v$  along the edge  $\{u, v\}$ , and denote it by  $I_i \leftrightarrow I_j$ . We note that the tokens are unlabeled, while the vertices in a graph are labeled. We sometimes omit to say the vertex on which a token is placed, and simply say a token in an independent set  $I$ .

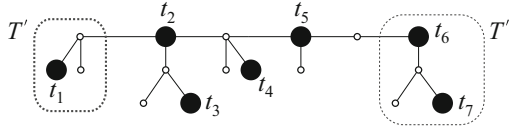
A *reconfiguration sequence* between two independent sets  $I_1$  and  $I_\ell$  of  $G$  is a sequence  $\langle I_1, I_2, \dots, I_\ell \rangle$  of independent sets of  $G$  such that  $I_{i-1} \leftrightarrow I_i$  for  $i = 2, 3, \dots, \ell$ . We sometimes write  $I \in \mathcal{S}$  if an independent set  $I$  of  $G$  appears in the reconfiguration sequence  $\mathcal{S}$ . We write  $I_1 \overset{G}{\rightsquigarrow} I_\ell$  if there exists a reconfiguration sequence  $\mathcal{S}$  between  $I_1$  and  $I_\ell$  such that all independent sets  $I \in \mathcal{S}$  satisfy  $I \subseteq V(G)$ ; we here define the notation emphasized with the graph  $G$ , because we will apply this notation to a subgraph of  $G$ . The *length* of a reconfiguration sequence  $\mathcal{S}$  is defined as the number of independent sets contained in  $\mathcal{S}$ . For example, the length of the reconfiguration sequence in Fig. 1 is 5.

Given two independent sets  $I_b$  and  $I_r$  of a graph  $G$ , the SLIDING TOKEN problem is to determine whether  $I_b \overset{G}{\rightsquigarrow} I_r$  or not. We may assume without loss of generality that  $|I_b| = |I_r|$ ; otherwise the answer is clearly “no.” Note that SLIDING TOKEN is a decision problem asking for the existence of a reconfiguration sequence between  $I_b$  and  $I_r$ , and hence it does not ask for an actual reconfiguration sequence. We always denote by  $I_b$  and  $I_r$  the *initial* and *target* independent sets of  $G$ , respectively.

**3 Algorithm for Trees**

In this section, we give the main result of this paper.

**Theorem 1.** *For a tree  $T$  with  $n$  vertices, the SLIDING TOKEN problem can be solved in  $O(n^2)$  time.*



**Fig. 4.** An independent set  $I$  of a tree  $T$ , where  $t_1, t_2, t_3, t_4$  are  $(T, I)$ -rigid tokens and  $t_5, t_6, t_7$  are  $(T, I)$ -movable tokens. Token  $t_1$  is  $(T', I \cap T')$ -movable for the subtree  $T'$ , and tokens  $t_6$  and  $t_7$  are  $(T'', I \cap T'')$ -rigid for the subtree  $T''$ .

As a proof of Theorem 1, we give an  $O(n^2)$ -time algorithm which simply solves SLIDING TOKEN for a tree with  $n$  vertices. In Section 3.3 we will show that an actual reconfiguration sequence can be obtained for a yes-instance on trees, and we will estimate its length.

### 3.1 Rigid Tokens

In this subsection, we formally define the concept of rigid tokens, and give their nice characterization.

Let  $T$  be a tree, and let  $I$  be an independent set of  $T$ . We say that a token on a vertex  $v \in I$  is  $(T, I)$ -rigid if  $v \in I'$  holds for any independent set  $I'$  of  $T$  such that  $I \overset{T}{\rightsquigarrow} I'$ . Conversely, if a token on a vertex  $v \in I$  is not  $(T, I)$ -rigid, then it is  $(T, I)$ -movable; in other words, there exists an independent set  $I'$  such that  $v \notin I'$  and  $I \overset{T}{\rightsquigarrow} I'$ . For example, in Fig. 4, the tokens  $t_1, t_2, t_3, t_4$  are  $(T, I)$ -rigid, while the tokens  $t_5, t_6, t_7$  are  $(T, I)$ -movable. Note that, even though  $t_6$  and  $t_7$  cannot be slid to any neighbor in  $I$ , we can slide them after sliding  $t_5$  downward.

We then extend the concept of rigid/movable tokens to subtrees of  $T$ . For any subtree  $T'$  of  $T$ , we denote simply  $I \cap T' = I \cap V(T')$ . Then, a token on a vertex  $v \in I \cap T'$  is  $(T', I \cap T')$ -rigid if  $v \in J$  holds for any independent set  $J$  of  $T'$  such that  $I \cap T' \overset{T'}{\rightsquigarrow} J$ ; otherwise the token is  $(T', I \cap T')$ -movable. For example, in Fig. 4, the token  $t_1$  is  $(T', I \cap T')$ -movable even though it is  $(T, I)$ -rigid in the whole tree  $T$ , while tokens  $t_6$  and  $t_7$  are  $(T'', I \cap T'')$ -rigid even though they are  $(T, I)$ -movable in  $T$ . Note that, since independent sets are restricted only to the subtree  $T'$ , we cannot use any vertex (and hence any edge) in  $T \setminus T'$  during the reconfiguration. Furthermore, the vertex-subset  $J \cup (I \cap (T \setminus T'))$  does not necessarily form an independent set of the whole tree  $T$ .

We now give our first key lemma, which gives a characterization of rigid tokens. (See also Fig. 5(a) for the claim (b) below.)

**Lemma 1.** *Let  $I$  be an independent set of a tree  $T$ , and let  $u$  be a vertex in  $I$ .*

- (a) *Suppose that  $|V(T)| = |\{u\}| = 1$ . Then, the token on  $u$  is  $(T, I)$ -rigid.*
- (b) *Suppose that  $|V(T)| \geq 2$ . Then, a token on  $u$  is  $(T, I)$ -rigid if and only if, for all neighbors  $v \in N(T, u)$ , there exists a vertex  $w \in I \cap N(T_v^u, v)$  such that the token on  $w$  is  $(T_w^v, I \cap T_w^v)$ -rigid.*





**Fig. 5.** (a) A  $(T, I)$ -rigid token on  $u$ , and (b) a  $(T, I)$ -movable token on  $u$

*Proof.* Obviously, the claim (a) holds. In the following, we thus assume that  $|V(T)| \geq 2$  and prove the claim (b).

We first show the if-part. Suppose that, for all neighbors  $v \in N(T, u)$ , there exists a vertex  $w \in I \cap N(T_v^u, v)$  such that the token on  $w$  is  $(T_w^v, I \cap T_w^v)$ -rigid. (See Fig. 5(a).) Then, we will prove that the token  $t$  on  $u$  is  $(T, I)$ -rigid. Since we can slide a token only along an edge of  $T$ , if  $t$  is not  $(T, I)$ -rigid (and hence is  $(T, I)$ -movable), then it must be slid to some neighbor  $v \in N(T, u)$ . By the assumption,  $v$  is adjacent with another token  $t'$  placed on  $w \in I \cap N(T_v^u, v)$ , and hence we first have to slide  $t'$  to one of its neighbors other than  $v$ . However, this is impossible since the token  $t'$  on  $w$  is assumed to be  $(T_w^v, I \cap T_w^v)$ -rigid and hence  $w \in J$  holds for any independent set  $J$  of  $T_w^v$  such that  $I \cap T_w^v \overset{T_w^v}{\rightsquigarrow} J$ . We can thus conclude that  $t$  is  $(T, I)$ -rigid.

We then show the only-if-part by taking a contrapositive. Suppose that  $u$  has a neighbor  $v \in N(T, u)$  such that either  $I \cap N(T_v^u, v) = \emptyset$  or all tokens on  $w \in I \cap N(T_v^u, v)$  are  $(T_w^v, I \cap T_w^v)$ -movable. (See Fig. 5(b).) Then, we will prove that the token  $t$  on  $u$  is  $(T, I)$ -movable; in particular, we can slide  $t$  from  $u$  to  $v$ . Since any token  $t'$  on a vertex  $w \in I \cap N(T_v^u, v)$  is  $(T_w^v, I \cap T_w^v)$ -movable, we can slide  $t'$  to some vertex in  $T_w^v$  via a reconfiguration sequence  $\mathcal{S}_w$  in  $T_w^v$ . Recall that only the vertex  $v$  is adjacent with a vertex in  $T_w^v$  and  $v \notin I$ . Therefore,  $\mathcal{S}_w$  can be naturally extended to a reconfiguration sequence  $\mathcal{S}$  in the whole tree  $T$  such that  $I' \cap (T \setminus T_w^v) = I \cap (T \setminus T_w^v)$  holds for any independent set  $I' \in \mathcal{S}$  of  $T$ . Apply this process to all tokens on vertices in  $I \cap N(T_v^u, v)$ , and obtain an independent set  $I''$  of  $T$  such that  $I'' \cap N(T_v^u, v) = \emptyset$ . Then, we can slide the token  $t$  on  $u$  to  $v$ . Thus,  $t$  is  $(T, I)$ -movable.  $\square$

Lemma 1 implies that we can check whether one token in an independent set  $I$  of a tree  $T$  is  $(T, I)$ -rigid or not in linear time.

**Lemma 2.** *Given a tree  $T$  with  $n$  vertices, an independent set  $I$  of  $T$ , and a vertex  $u \in I$ , it can be decided in  $O(n)$  time whether the token on  $u$  is  $(T, I)$ -rigid.*

The following lemma is useful for our algorithm in Section 3.2.

**Lemma 3.** *Let  $I$  be an independent set of a tree  $T$  such that all tokens are  $(T, I)$ -movable, and let  $v$  be a vertex such that  $v \notin I$ . Then, there exists at most one neighbor  $w \in I \cap N(T, v)$  such that the token on  $w$  is  $(T_w^v, I \cap T_w^v)$ -rigid.*

*Proof.* Suppose for a contradiction that there exist two neighbors  $w$  and  $w'$  in  $I \cap N(T, v)$  such that the tokens on  $w$  and  $w'$  are  $(T_w^v, I \cap T_w^v)$ -rigid and

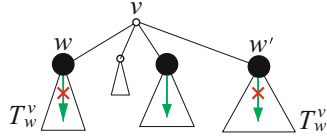


Fig. 6. Illustration for Lemma 3

$(T_w^v, I \cap T_w^v)$ -rigid, respectively. (See Fig. 6.) Since the token  $t$  on  $w$  is  $(T_w^v, I \cap T_w^v)$ -rigid but is  $(T, I)$ -movable, there is a reconfiguration sequence  $\mathcal{S}_t$  starting from  $I$  which slides  $t$  to  $v$ . However, before sliding  $t$  to  $v$ ,  $\mathcal{S}_t$  must slide the token  $t'$  on  $w'$  to some vertex in  $N(T_{w'}^v, w')$ . This contradicts the assumption that  $t'$  is  $(T_{w'}^v, I \cap T_{w'}^v)$ -rigid.  $\square$

### 3.2 Algorithm

In this subsection, we describe a quadratic-time algorithm to solve the SLIDING TOKEN problem on trees, and prove its correctness.

Let  $T$  be a tree with  $n$  vertices, and let  $I_b$  and  $I_r$  be two given independent sets of  $T$ . For an independent set  $I$  of  $T$ , we denote by  $R(I)$  the set of all vertices in  $I$  on which  $(T, I)$ -rigid tokens are placed.

**Step 1.** Compute  $R(I_b)$  and  $R(I_r)$  using Lemma 2. If  $R(I_b) \neq R(I_r)$ , then return “no”; otherwise go to Step 2.

**Step 2.** Delete the vertices in  $N[T, R(I_b)] = N[T, R(I_r)]$  from  $T$ , and obtain a forest  $F$  consisting of  $q$  trees  $T_1, T_2, \dots, T_q$ . If  $|I_b \cap T_j| = |I_r \cap T_j|$  holds for every  $j \in \{1, 2, \dots, q\}$ , then return “yes”; otherwise return “no.”

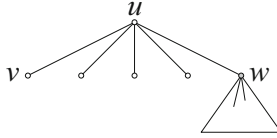
By Lemma 2 we can determine whether one token in an independent set  $I$  of  $T$  is  $(T, I)$ -rigid or not in  $O(n)$  time, and hence Step 1 can be done in time  $O(n) \times (|I_b| + |I_r|) = O(n^2)$ . Clearly, Step 2 can be done in  $O(n)$  time. Therefore, our algorithm above runs in  $O(n^2)$  time in total. In the remainder of this subsection, we thus prove the correctness of our algorithm.

We first show the correctness of Step 1.

**Lemma 4.** *Suppose that  $R(I_b) \neq R(I_r)$  for two given independent sets  $I_b$  and  $I_r$  of a tree  $T$ . Then, it is a no-instance.*

We then show the correctness of Step 2. We first claim that deleting the vertices with rigid tokens together with their neighbors does not affect the reconfigurability.

**Lemma 5.** *Suppose that  $R(I_b) = R(I_r)$  for two given independent sets  $I_b$  and  $I_r$  of a tree  $T$ , and let  $F$  be the forest obtained by deleting the vertices in  $N[T, R(I_b)] = N[T, R(I_r)]$  from  $T$ . Then,  $I_b \overset{T}{\longleftrightarrow} I_r$  if and only if  $I_b \cap F \overset{F}{\longleftrightarrow} I_r \cap F$ . Furthermore, all tokens in  $I_b \cap F$  are  $(F, I_b \cap F)$ -movable, and all tokens in  $I_r \cap F$  are  $(F, I_r \cap F)$ -movable.*



**Fig. 7.** A degree-1 vertex  $v$  of a tree  $T$  which is safe

Suppose that  $R(I_b) = R(I_r)$  for two given independent sets  $I_b$  and  $I_r$  of a tree  $T$ . Let  $F$  be the forest consisting of  $q$  trees  $T_1, T_2, \dots, T_q$ , which is obtained from  $T$  by deleting the vertices in  $N[T, R(I_b)] = N[T, R(I_r)]$ . Since we can slide a token only along an edge of  $F$ , we clearly have  $I_b \cap F \overset{F}{\rightsquigarrow} I_r \cap F$  if and only if  $I_b \cap T_j \overset{T_j}{\rightsquigarrow} I_r \cap T_j$  for all  $j \in \{1, 2, \dots, q\}$ . Furthermore, Lemma 5 implies that, for each  $j \in \{1, 2, \dots, q\}$ , all tokens in  $I_b \cap T_j$  are  $(T_j, I_b \cap T_j)$ -movable; similarly, all tokens in  $I_r \cap T_j$  are  $(T_j, I_r \cap T_j)$ -movable.

We now give our second key lemma, which completes the correctness proof of our algorithm.

**Lemma 6.** *Let  $I_b$  and  $I_r$  be two independent sets of a tree  $T$  such that all tokens in  $I_b$  and  $I_r$  are  $(T, I_b)$ -movable and  $(T, I_r)$ -movable, respectively. Then,  $I_b \overset{T}{\rightsquigarrow} I_r$  if and only if  $|I_b| = |I_r|$ .*

The only-if-part is trivial, and hence we prove the if-part. In our proof, we do *not* reconfigure  $I_b$  into  $I_r$  directly, but reconfigure both  $I_b$  and  $I_r$  into some independent set  $I^*$  of  $T$ .

We say that a degree-1 vertex  $v$  of  $T$  is *safe* if its unique neighbor  $u$  has at most one neighbor  $w$  of degree more than one. (See Fig. 7.) Note that any tree has at least one safe degree-1 vertex.

As the first step of the if-part proof, we give the following lemma.

**Lemma 7.** *Let  $I$  be an independent set of a tree  $T$  such that all tokens in  $I$  are  $(T, I)$ -movable, and let  $v$  be a safe degree-1 vertex of  $T$ . Then, there exists an independent set  $I'$  such that  $v \in I'$  and  $I \overset{T}{\rightsquigarrow} I'$ .*

*Proof.* Suppose that  $v \notin I$ ; otherwise the lemma clearly holds. We will show that one of the closest tokens from  $v$  can be slid to  $v$ . Let  $M = \{w \in I \mid \text{dist}(v, w) = \min_{x \in I} \text{dist}(v, x)\}$ . Let  $w$  be an arbitrary vertex in  $M$ , and let  $P = (p_0 = v, p_1, \dots, p_\ell = w)$  be the  $vw$ -path in  $T$ . (See Fig. 8.) If  $\ell = 1$  and hence  $p_1 \in I$ , then we can simply slide the token on  $p_1$  to  $v$ . Thus, we may assume that  $\ell \geq 2$ .

We note that no token is placed on the vertices  $p_0, \dots, p_{\ell-1}$  and the neighbors of  $p_0, \dots, p_{\ell-2}$ , because otherwise the token on  $w$  is not closest to  $v$ . Let  $M' = M \cap N(T, p_{\ell-1})$ . Since  $p_{\ell-1} \notin I$ , by Lemma 3 there exists at most one vertex  $w' \in M'$  such that the token on  $w'$  is  $(T_{w'}^{p_{\ell-1}}, I \cap T_{w'}^{p_{\ell-1}})$ -rigid. We choose such a vertex  $w'$  if it exists, otherwise choose an arbitrary vertex in  $M'$  and regard it as  $w'$ .

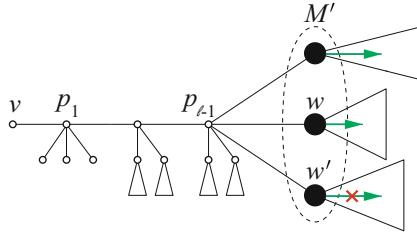


Fig. 8. Illustration for Lemma 7

Since all tokens on the vertices  $w''$  in  $M' \setminus \{w'\}$  are  $(T_{w''}^{p_{l-1}}, I \cap T_{w''}^{p_{l-1}})$ -movable, we first slide the tokens on  $w''$  to some vertices in  $T_{w''}^{p_{l-1}}$ . Then, we can slide the token on  $w'$  to  $v$  along the path  $P$ . In this way, we can obtain an independent set  $I'$  such that  $v \in I'$  and  $I \overset{T}{\rightsquigarrow} I'$ .  $\square$

We then prove that deleting a safe degree-1 vertex with a token does not affect the movability of the other tokens.

**Lemma 8.** *Let  $v$  be a safe degree-1 vertex of a tree  $T$ , and let  $\bar{T}$  be the subtree of  $T$  obtained by deleting  $v$ , its unique neighbor  $u$ , and the resulting isolated vertices. Let  $I$  be an independent set of  $T$  such that  $v \in I$  and all tokens are  $(T, I)$ -movable. Then, all tokens in  $I \setminus \{v\}$  are  $(\bar{T}, I \setminus \{v\})$ -movable.*

In Lemma 8, note that the token on  $v$  is  $(T_v^u, I \cap T_v^u)$ -rigid since  $T_v^u$  consists of a single vertex  $v$ . Therefore, no token is placed on degree-1 neighbors of  $u$  other than  $v$ , because otherwise it contradicts to Lemma 3; recall that all tokens in  $I$  are assumed to be  $(T, I)$ -movable.

**Proof of the if-part of Lemma 6**

We prove the if-part of the lemma by the induction on the number of tokens  $|I_b| = |I_r|$ . The lemma clearly holds for any tree  $T$  if  $|I_b| = |I_r| = 1$ , because  $T$  has only one token and hence we can slide it along the unique path in  $T$ .

We choose an arbitrary safe degree-1 vertex  $v$  of a tree  $T$ , whose unique neighbor is  $u$ . Since all tokens in  $I_b$  are  $(T, I_b)$ -movable, by Lemma 7 we can obtain an independent set  $I'_b$  of  $T$  such that  $v \in I'_b$  and  $I_b \overset{T}{\rightsquigarrow} I'_b$ . By Lemma 8 all tokens in  $I'_b \setminus \{v\}$  are  $(\bar{T}, I'_b \setminus \{v\})$ -movable, where  $\bar{T}$  is the subtree defined in Lemma 8. Similarly, we can obtain an independent set  $I'_r$  of  $T$  such that  $v \in I'_r$ ,  $I_r \overset{T}{\rightsquigarrow} I'_r$  and all tokens in  $I'_r \setminus \{v\}$  are  $(\bar{T}, I'_r \setminus \{v\})$ -movable. Apply the induction hypothesis to the pair of independent sets  $I'_b \setminus \{v\}$  and  $I'_r \setminus \{v\}$  of  $\bar{T}$ . Then, we have  $I'_b \setminus \{v\} \overset{\bar{T}}{\rightsquigarrow} I'_r \setminus \{v\}$ . Recall that both  $u \notin I'_b$  and  $u \notin I'_r$  hold, and  $u$  is the unique neighbor of  $v$  in  $T$ . Therefore, we can extend the reconfiguration sequence in  $\bar{T}$  between  $I'_b \setminus \{v\}$  and  $I'_r \setminus \{v\}$  to a reconfiguration sequence in  $T$  between  $I'_b$  and  $I'_r$ . We thus have  $I_b \overset{T}{\rightsquigarrow} I_r$ .

This completes the proof of Lemma 6, and hence completes the proof of Theorem 1.  $\square$



Fig. 9. No-instance for an interval graph such that all tokens are movable

### 3.3 Length of Reconfiguration Sequence

In this subsection, we show that an actual reconfiguration sequence can be found for a yes-instance on trees, by implementing our proofs in Section 3.2. Furthermore, the length of the obtained reconfiguration sequence is at most quadratic.

**Theorem 2.** *Let  $I_b$  and  $I_r$  be two independent sets of a tree  $T$  with  $n$  vertices. If  $I_b \overset{T}{\rightsquigarrow} I_r$ , then there exists a reconfiguration sequence of length  $O(n^2)$  between  $I_b$  and  $I_r$ , and it can be found in  $O(n^2)$  time.*

It is interesting that there exists an infinite family of instances on paths for which any reconfiguration sequence requires  $\Omega(n^2)$  length, where  $n$  is the number of vertices. For example, consider a path  $(v_1, v_2, \dots, v_{8k})$  with  $n = 8k$  vertices for any positive integer  $k$ , and let  $I_b = \{v_1, v_3, v_5, \dots, v_{2k-1}\}$  and  $I_r = \{v_{6k+2}, v_{6k+4}, \dots, v_{8k}\}$ . In this yes-instance, any token must be slid  $\Theta(n)$  times, and hence any reconfiguration sequence requires  $\Theta(n^2)$  length to slide them all.

## 4 Concluding Remarks

In this paper, we have developed an  $O(n^2)$ -time algorithm to solve the SLIDING TOKEN problem for trees with  $n$  vertices, based on a simple but non-trivial characterization of rigid tokens. We have shown that there exists a reconfiguration sequence of length  $O(n^2)$  for any yes-instance on trees, and it can be found in  $O(n^2)$  time; while there exists an infinite family of instances on paths for which any reconfiguration sequence requires  $\Omega(n^2)$  length.

Recently, we have improved the running time of our algorithm [7]: we proposed a linear-time algorithm which simply decides whether  $I_b \overset{T}{\rightsquigarrow} I_r$  or not, for two given independent sets  $I_b$  and  $I_r$  of a tree  $T$ .

The complexity status of SLIDING TOKEN remains open for chordal graphs and interval graphs. Interestingly, these graphs have no-instances such that all tokens are movable. (See Fig. 9 for example.)

**Acknowledgments.** The authors thank anonymous referees for their helpful suggestions. This work is supported in part by NSF grant CCF-1161626 and DARPA/AFOSR grant FA9550-12-1-0423 (E.D. Demaine), and by JSPS KAKENHI 25106504 and 25330003 (T. Ito), 25104521, 26540005 and 26540005 (H. Ono) and 26330009 (R. Uehara).

## References

1. Bodlaender, H.L.: A partial  $k$ -arboretum of graphs with bounded treewidth. *Theoretical Computer Science* **209**, 1–45 (1998)
2. Bonamy, M., Johnson, M., Lignos, I., Patel, V., Paulusma, D.: Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *J. Combinatorial Optimization* **27**, 132–143 (2014)
3. Bonsma, P.: The complexity of rerouting shortest paths. *Theoretical Computer Science* **510**, 1–12 (2013)
4. Bonsma, P.: Independent set reconfiguration in cographs. To appear in WG 2014, [arXiv:1402.1587](https://arxiv.org/abs/1402.1587) (2014)
5. Bonsma, P., Cereceda, L.: Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science* **410**, 5215–5226 (2009)
6. Bonsma, P., Kamiński, M., Wrochna, M.: Reconfiguring independent sets in claw-free graphs. In: Ravi, R., Gørtz, I.L. (eds.) SWAT 2014. LNCS, vol. 8503, pp. 86–97. Springer, Heidelberg (2014)
7. Demaine, E.D., Demaine, M.L., Fox-Epstein, E., Hoang, D.A., Ito, T., Ono, H., Otachi, Y., Uehara, R., Yamada, T.: Linear-time algorithm for sliding tokens on trees [arXiv:1406.6576](https://arxiv.org/abs/1406.6576) (2014)
8. Gopalan, P., Kolaitis, P.G., Maneva, E.N., Papadimitriou, C.H.: The connectivity of Boolean satisfiability: computational and structural dichotomies. *SIAM J. Computing* **38**, 2330–2355 (2009)
9. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science* **343**, 72–96 (2005)
10. Hearn, R.A., Demaine, E.D.: *Games, Puzzles, and Computation*. A K Peters (2009)
11. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C.H., Sideri, M., Uehara, R., Uno, Y.: On the complexity of reconfiguration problems. *Theoretical Computer Science* **412**, 1054–1065 (2011)
12. Ito, T., Kamiński, M., Ono, H., Suzuki, A., Uehara, R., Yamanaka, K.: On the parameterized complexity for token jumping on graphs. In: Gopal, T.V., Agrawal, M., Li, A., Cooper, S.B. (eds.) TAMC 2014. LNCS, vol. 8402, pp. 341–351. Springer, Heidelberg (2014)
13. Ito, T., Kawamura, K., Ono, H., Zhou, X.: Reconfiguration of list  $L(2, 1)$ -labelings in a graph. *Theoretical Computer Science* **544**, 84–97 (2014)
14. Kamiński, M., Medvedev, P., Milanič, M.: Shortest paths between shortest paths. *Theoretical Computer Science* **412**, 5205–5210 (2011)
15. Kamiński, M., Medvedev, M., Milanič, M.: Complexity of independent set reconfigurability problems. *Theoretical Computer Science* **439**, 9–15 (2012)
16. Makino, K., Tamaki, S., Yamamoto, M.: An exact algorithm for the Boolean connectivity problem for  $k$ -CNF. *Theoretical Computer Science* **412**, 4613–4618 (2011)
17. Mouawad, A.E., Nishimura, N., Raman, V., Simjour, N., Suzuki, A.: On the parameterized complexity of reconfiguration problems. In: Gutin, G., Szeider, S. (eds.) IPEC 2013. LNCS, vol. 8246, pp. 281–294. Springer, Heidelberg (2013)
18. Mouawad, A.E., Nishimura, N., Raman, V., Wrochna, M.: Reconfiguration over tree decompositions [arXiv:1405.2447](https://arxiv.org/abs/1405.2447)
19. van den Heuvel, J.: The complexity of change. *Surveys in Combinatorics 2013*, London Mathematical Society Lecture Notes Series 409 (2013)
20. Wrochna, M.: Reconfiguration in bounded bandwidth and treedepth [arXiv:1405.0847](https://arxiv.org/abs/1405.0847) (2014)

# Minimal Obstructions for Partial Representations of Interval Graphs

Pavel Klavík<sup>1</sup>(✉) and Maria Saumell<sup>2</sup>

<sup>1</sup> Computer Science Institute, Charles University in Prague, Prague, Czech Republic  
klavik@iuuk.mff.cuni.cz

<sup>2</sup> Department of Mathematics and European Centre of Excellence NTIS,  
University of West Bohemia, Pilsen, Czech Republic  
saumell@kma.zcu.cz

**Abstract.** *Interval graphs* are intersection graphs of closed intervals. A generalization of recognition called *partial representation extension* was introduced recently. The input gives an interval graph with a *partial representation* specifying some pre-drawn intervals. We ask whether the remaining intervals can be added to create an *extending representation*.

In this paper, we characterize the *minimal obstructions* which make a partial representation non-extendible. This generalizes Lekkerkerker and Boland's characterization of minimal forbidden induced subgraphs of interval graphs. Each minimal obstruction consists of a forbidden induced subgraph together with at most four pre-drawn intervals. A Helly-type result follows: A partial representation is extendible if and only if every quadruple of pre-drawn intervals is extendible by itself. Our characterization leads to the first polynomial-time certifying algorithm for partial representation extension of intersection graphs.

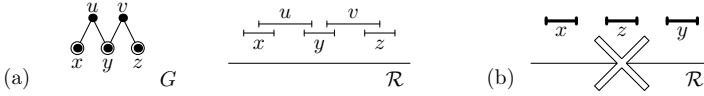
## 1 Introduction

The main motivation for graph drawing and geometric representations is finding ways to visualize some given data efficiently. We study famous *interval graphs* (INT), introduced by Hájos [10] in 1957. An *interval representation*  $\mathcal{R}$  is a collection of closed intervals  $\{\langle x \rangle : x \in V(G)\}$  where  $\langle x \rangle \cap \langle y \rangle \neq \emptyset$  if and only if  $xy \in E(G)$ ; so edges are encoded by intersections. A graph is an interval graph if it has an interval representation; see Fig. 1a.

Interval graphs have many applications [3, 12, 21] and nice theoretical properties. They are perfect and closely related to path-width decompositions. Fulkerson and Gross [9] characterized them by consecutive orderings of maximal cliques (see Section 3 for details). This led Booth and Lueker [5] to invent PQ-trees, which are an efficient data structure to deal with consecutive orderings, can recognize interval graphs in linear time, and have many other applications.

---

For the full version of this paper, see [arXiv:1406.6228](https://arxiv.org/abs/1406.6228). This work was initiated during a EUROCORES Short Term Visit at ULB in Brussels. The first author is supported by CE-ITI (P202/12/G061 of GAČR) and Charles University as GAUK 196213; the second author by the project NEXLIZ - CZ.1.07/2.3.00/30.0038, which is co-financed by the European Social Fund and the state budget of the Czech Republic, and by ESF EuroGIGA project ComPoSe as F.R.S.-FNRS - EUROGIGA NR 13604.



**Fig. 1.** (a) An interval graph  $G$  with one of its interval representations  $\mathcal{R}$ . (b) A non-extendible partial representation  $\mathcal{R}'$ . Pre-drawn intervals are bold in all figures.

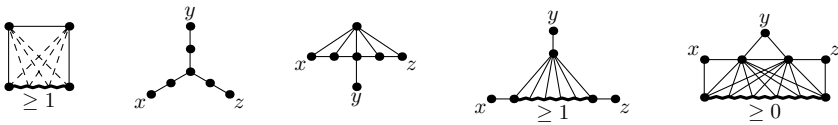
*Chordal graphs* (CHOR) are graphs containing no induced cycle of length four or more, or alternatively intersection graphs of subtrees of trees. Given a graph, three vertices form an *asteroidal triple* if there exists a path between every pair of them avoiding the neighborhood of the third vertex. *Asteroidal triple-free graphs* (AT-FREE) are graphs containing no asteroidal triple. Lekkerkerker and Boland [19] characterized interval graphs as  $\text{INT} = \text{CHOR} \cap \text{AT-FREE}$ . They described this characterization using minimal forbidden induced subgraphs, which we call *Lekkerkerker-Boland obstructions* (LB); see Fig. 2.

**Partial Representation Extension.** This problem was introduced by Klavík et al. [17]. For interval graphs, a *partial representation*  $\mathcal{R}'$  is an interval representation  $\{\langle x \rangle' : x \in V(G')\}$  of an induced subgraph  $G'$  of  $G$ . The vertices/intervals of  $G'$  are called *pre-drawn*. A representation  $\mathcal{R}$  of  $G$  *extends*  $\mathcal{R}'$  if and only if it assigns the same intervals to the vertices of  $G'$ , i.e.,  $\langle x \rangle = \langle x \rangle'$  for every  $x \in V(G')$ . For an example, see Fig. 1b.

**Problem:** Partial Representation Extension – REPEXT(INT)  
**Input:** A graph  $G$  and a partial representation  $\mathcal{R}'$  of  $G'$ .  
**Output:** Is there an interval representation of  $G$  extending  $\mathcal{R}'$ ?

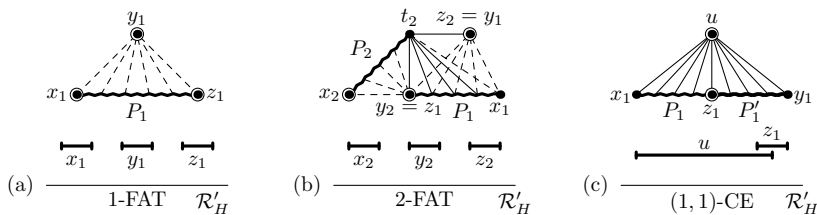
For interval graphs, the partial representation extension problem was solved in  $\mathcal{O}(n^2)$  time in [17], and currently there are two different linear-time algorithms for this problem [4, 16]. A linear-time algorithm for proper interval graphs and an almost quadratic-time algorithm for unit interval graphs are given in [14]. Polynomial-time algorithms are further known for circle graphs [7], and permutation and function graphs [13]. The partial representation extension problems for chordal graphs and contact representations of planar graphs are NP-hard [6, 15].

Partially embedded planar graphs can be extended in linear-time [1]. Even though every planar graph has a straight-line embedding, extension of such embeddings is NP-hard [20]. Kuratowski’s characterization of minimal forbidden minors was extended to partially embedded planar graphs by Jelínek et al. [11]. Our research has a similar spirit as this last result.



**Fig. 2.** Five types of LB obstructions which are minimal forbidden induced subgraphs of INT. The leftmost obstruction is an induced cycle of length four or more. The remaining obstructions are minimal asteroidal triples  $(x, y, z)$  which are chordal graphs. In all figures, curly lines denote induced paths and dashed edges are non-edges.





**Fig. 3.** Three examples of minimal obstructions. Each of them consists of a graph  $H$  and a non-extendible partial representation  $\mathcal{R}'_H$ .

**Our Results.** In this paper, we generalize the characterization of Lekkerkerker and Boland [19] to describe extendible partial representations. The main class of obstructions for extendability, called  $k$ -FAT obstructions, has three wrongly ordered disjoint pre-drawn intervals  $x_k, y_k$  and  $z_k$ ; see Fig. 3a and b. There are seven other infinite classes of obstructions which are derived from  $k$ -FAT obstructions by adding a few vertices and having different pre-drawn vertices. The last infinite class of  $(k, \ell)$ -CE obstructions consists of a  $k$ -FAT obstruction glued to an  $\ell$ -FAT obstruction; see Fig. 3c. See Section 2 for definitions.

**Theorem 1.1.** *A partial representation  $\mathcal{R}'$  of  $G$  is extendible if and only if  $G$  and  $\mathcal{R}'$  contain no LB, SE,  $k$ -FAT,  $k$ -BI,  $k$ -FS,  $k$ -EFS,  $k$ -FB,  $k$ -FDS,  $k$ -EFDS,  $k$ -FNS and  $(k, \ell)$ -CE obstructions.*

We use the characterization of extendible partial representations of [16] by maximal cliques, which we describe in Section 4. We extend these structural results. In Section 5, we get forbidden configurations of maximal cliques which we translate into minimal obstructions. We get the following Helly-type result:

**Corollary 1.2.** *A partial representation is extendible if and only if every four pre-drawn intervals are extendible by themselves.*

All previously known algorithms for partial representation extension are able to certify solvable instances by giving an extending representation. Using our minimal obstructions, we construct the first algorithm for partial representation extension certifying also non-extendible partial representations.<sup>1</sup>

**Corollary 1.3.** *There exists an  $\mathcal{O}(nm)$  certifying algorithm for the partial representation extension problem, where  $n$  is the number of vertices and  $m$  is the number of edges of the input graph. If the answer is “yes”, it outputs an extending representation. If the answer is “no”, it detects one of the minimal obstructions.*

**Notation.** For a graph  $G$ , we denote by  $V(G)$  its vertices and by  $E(G)$  its edges. We denote the *closed neighborhood* of  $x$  by  $N[x]$ . Maximal cliques are denoted by the letters  $a$  to  $f$ , and vertices by the remaining letters. For an interval  $\langle x \rangle$ , we denote its left endpoint by  $\ell(x)$  and its right endpoint by  $r(x)$ . Omitted details and proofs can be found in the full version, see arXiv:1406.6228.

<sup>1</sup> Formally speaking, the polynomial-time algorithm of [16] certifies non-extendible instances by outputting “no” and by the proof of its correctness. Our algorithm outputs a simple proof that a given partial representation is non-extendible. This proof can be independently verified which is desirable.

## 2 Definition of Minimal Obstructions

Every obstruction consists of a graph  $H$  and a non-extendible partial representation  $\mathcal{R}'_H$ . We describe  $H$  using some vertices and induced paths. For inner vertices of the induced paths, we specify their adjacencies with the remainder of  $H$ . Since these induced paths do not have fixed lengths, each description with an induced path defines an infinite class of forbidden subgraphs  $H$ .

An obstruction consisting of  $H$  and  $\mathcal{R}'_H$  is *contained* in  $G$  and  $\mathcal{R}'$  if (i)  $H$  is an induced subgraph of  $G$ , (ii) the pre-drawn vertices of  $H$  are mapped to pre-drawn vertices of  $G$ , and (iii) the endpoints in  $\mathcal{R}'_H$  are ordered the same as the endpoints of the corresponding pre-drawn vertices in  $\mathcal{R}'$ . An obstruction is *minimal* if  $\mathcal{R}'_H$  becomes extendible when any vertex or induced path is removed, or some pre-drawn interval is made *free* by removing it from the partial representation  $\mathcal{R}'_H$ .

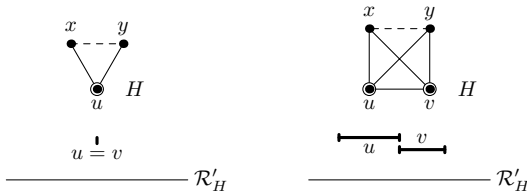


Fig. 4. Two SE obstructions. On the left we have  $u = v$ , while on the right  $u \neq v$ .

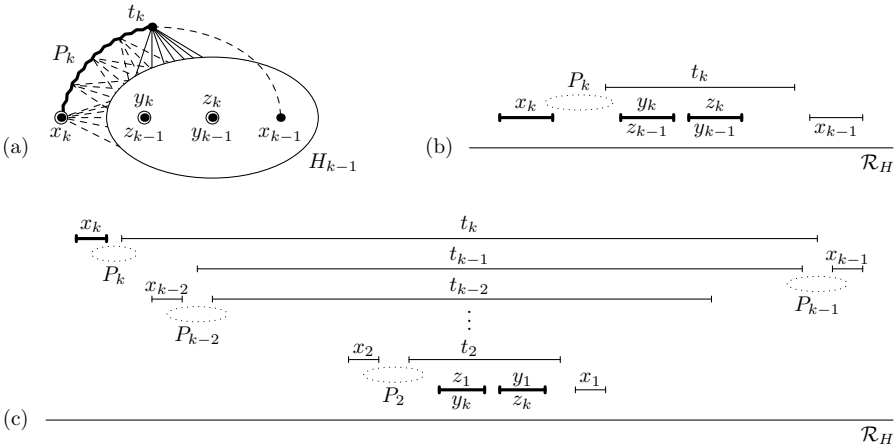


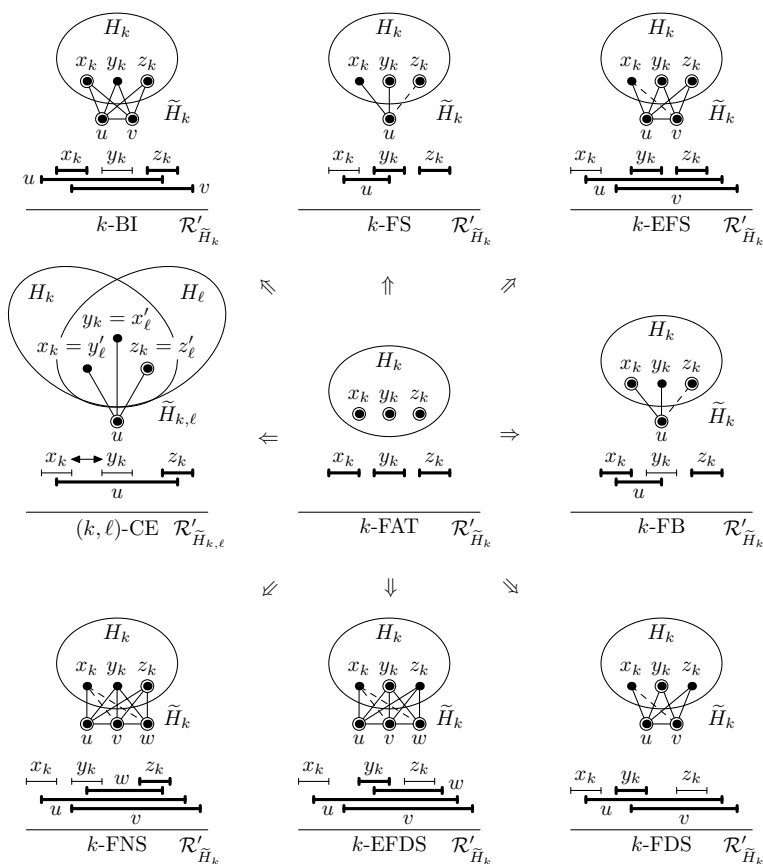
Fig. 5. (a) A  $k$ -FAT obstruction is created from a  $(k - 1)$ -FAT obstruction by adding vertices  $x_k$  and  $t_k$  connected by an induced path  $P_k$ . In other words, a  $k$ -FAT obstruction consists of vertices  $x_1, \dots, x_k, t_2, \dots, t_k, y_k, z_k$ , and induced paths  $P_1, \dots, P_k$ . The adjacencies are defined inductively as above. (b) In every representation  $\mathcal{R}_H$ , the pre-drawn interval  $\langle x_k \rangle'$  together with  $P_k$  and  $t_k$  force  $x_{k-1}$  to be placed on the right of  $z_k$ . Therefore, the induced  $(k - 1)$ -FAT obstruction is forced. (c) The global zig-zag pattern forced by a  $k$ -FAT obstruction, with  $k$  nested levels going across  $y_k$  and  $z_k$ . It is an obstruction, since  $P_1$  with all inner vertices non-adjacent to  $y_1$  cannot be placed.

Aside from LB obstructions of [19] with  $\mathcal{R}'_H = \emptyset$ , we have ten other classes.

**SE obstructions.** These are two simple *shared endpoint obstructions* which deal with shared endpoints in  $\mathcal{R}'$ . They are depicted in Fig. 4.

**$k$ -FAT obstructions.** The class of *forced asteroidal triple obstructions* consists of obstructions containing three pre-drawn vertices and  $k$  induced paths  $P_1, \dots, P_k$ . The obstructions are defined inductively.

The 1-FAT obstruction consists of three pre-drawn non-adjacent vertices  $x_1, y_1$  and  $z_1$  such that  $y_1$  is between  $x_1$  and  $z_1$ . Further,  $x_1$  and  $z_1$  are connected by an induced path  $P_1$  and  $y_1$  is non-adjacent to the inner vertices of  $P_1$ . See Fig. 3b. The  $k$ -FAT obstruction is defined as follows. Let  $H_{k-1}$  be a graph for  $(k-1)$ -FAT. To get  $k$ -FAT, we add to  $H_{k-1}$  two vertices  $x_k$  and  $t_k$  connected by an induced path  $P_k$ . We name  $y_k = z_{k-1}$  and  $z_k = y_{k-1}$ . Concerning edges,



**Fig. 6.** The classes of obstructions derived from  $k$ -FAT: *blocked intersection obstructions* ( $k$ -BI), *forced side obstructions* ( $k$ -FS), *extended forced side obstructions* ( $k$ -EFS), *forced betweenness obstructions* ( $k$ -FB), *forced different sides obstructions* ( $k$ -FDS), *extended forced different sides obstructions* ( $k$ -EFDS), *forced nested side obstructions* ( $k$ -FNS), and *covered endpoint obstructions* ( $(k, \ell)$ -CE).

$t_k$  is adjacent to all vertices of  $H_{k-1}$ , except for  $x_{k-1}$ . All vertices of  $H_{k-1}$  are non-adjacent to  $x_k$  and to the inner vertices of  $P_k$ . See Fig. 5a.

The  $k$ -FAT obstruction has three pre-drawn vertices  $x_k, y_k$  and  $z_k$  such that  $y_k$  is placed in  $\mathcal{R}'_{H_k}$  between  $x_k$  and  $z_k$ . The role of  $x_k, P_k$  and  $t_k$  is to force  $x_{k-1}$  to be placed on the other side of  $z_k = y_{k-1}$  than  $y_k = z_{k-1}$ , thus forcing the  $(k - 1)$ -FAT obstruction of  $H_{k-1}$ ; see Fig. 5b. The global structure forced by a  $k$ -FAT obstruction is depicted in Fig. 5c.

**Derived Classes.** The remaining eight classes, depicted in Fig. 6, are derived from  $k$ -FAT obstructions. Let  $H_k$  denote the graph of a  $k$ -FAT obstruction. Except for the last class  $(k, \ell)$ -CE, we create the graphs  $\tilde{H}_k$  of these obstructions by adding a few vertices to  $H_k$ . The more complex  $(k, \ell)$ -CE obstructions consist of two FAT obstructions glued together. If  $\langle x_k \rangle$  is on the left of  $\langle y_k \rangle$ , then this leads to a  $k$ -FAT obstruction; if  $\langle x_k \rangle$  is on the right of  $\langle y_k \rangle$ , then this leads to an  $\ell$ -FAT obstruction. The precise definitions are in the full version.

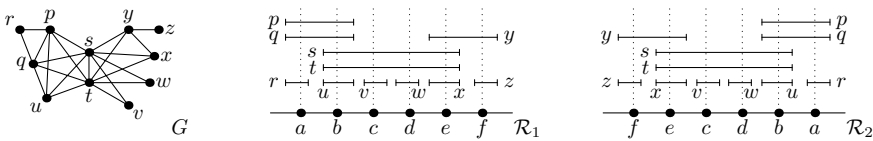
**Lemma 2.1.** *SE,  $k$ -FAT,  $k$ -BI,  $k$ -FS,  $k$ -EFS,  $k$ -FB,  $k$ -FDS,  $k$ -EFDS,  $k$ -FNS and  $(k, \ell)$ -CE obstructions are non-extendible and minimal.*

This implies the first part of Theorem 1.1, which states that if  $G$  and  $\mathcal{R}'$  contain one of the obstructions, then  $\mathcal{R}'$  is non-extendible. We establish the harder opposite implication in Section 5.

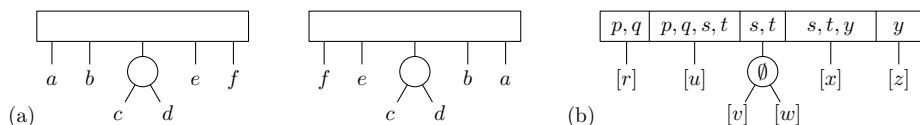
### 3 Maximal Cliques and MPQ-Trees

A linear ordering  $<$  of the maximal cliques of a graph is *consecutive* if, for each vertex, the maximal cliques containing this vertex appear consecutively. Fulkerson and Gross [9] proved that a graph is an interval graph if and only if there exists a consecutive ordering of its maximal cliques. The reason is that the intervals of every maximal clique  $a$  have a common intersection. Given a representation, we can pick one point for  $a$  called a *clique-point*  $cp(a)$ , and the left-to-right ordering of these clique-points gives  $<$ . See Fig. 7 for an example.

**PQ-trees.** Booth and Lueker [5] invented PQ-trees to work efficiently with consecutive orderings. A *PQ-tree*  $T$  is a rooted tree. Its leaves are in one-to-one correspondence with the maximal cliques of  $G$ . Its inner nodes are of two types: *P-nodes* and *Q-nodes*. Each P-node has at least two children, and each Q-node has at least three. Further, for every inner node, the ordering of its children is fixed. Every PQ-tree  $T$  represents one linear ordering  $<_T$  of the maximal cliques called the *frontier* of  $T$ , which is the ordering of the leaves from left to right.



**Fig. 7.** An interval graph  $G$  and two of its representations with different left-to-right orderings of the maximal cliques. Some choices of clique-points are depicted.



**Fig. 8.** (a) Two equivalent PQ-trees with frontiers  $a < b < c < d < e < f$  and  $f < e < c < d < b < a$ . (b) The MPQ-tree corresponding to the left PQ-tree.

Every PQ-tree  $T$  additionally represents other linear orderings. These orderings are frontiers of equivalent PQ-trees. A PQ-tree  $T'$  is *equivalent* to  $T$  if it can be constructed from  $T$  by a sequence of *equivalent transformations*, which can be of two types: (i) an arbitrary reordering of the children of a P-node, and (ii) a reversal of the order of the children of a Q-node. Fig. 8a depicts two equivalent PQ-trees corresponding to the interval graph from Fig. 7.

Booth and Lueker proved that, for every interval graph, there exists a unique PQ-tree representing all consecutive orderings of the maximal cliques. This tree describes all possible interval representations of this interval graph.

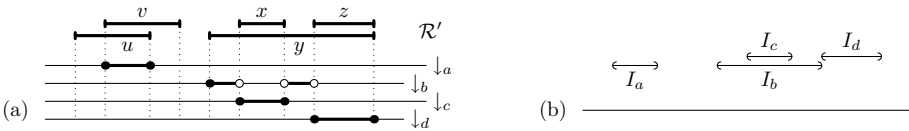
**MPQ-trees.** The *modified PQ-tree* (MPQ-tree), introduced by Korte and Möhring [18], gives information about the relation between the vertices of  $G$  and the structure of the PQ-tree. We note that the same idea is already present in the paper of Colbourn and Booth [8]. The MPQ-tree is an augmentation of the PQ-tree in which the nodes of  $T$  have assigned subsets of  $V(G)$  called *sections*. To a leaf representing a clique  $a$ , we assign one section  $s(a)$ . Similarly, to each P-node  $P$ , we assign one section  $s(P)$ . For a Q-node  $Q$  with subtrees  $T_1, \dots, T_n$ , we have  $n$  sections  $s_1(Q), \dots, s_n(Q)$ , each corresponding to one subtree.

The section  $s(a)$  has all vertices contained in the maximal clique  $a$  and in no other maximal clique. The section  $s(P)$  of a P-node  $P$  has all vertices that are contained in all maximal cliques of the subtree rooted at  $P$  and in no other maximal clique. Sections of Q-nodes are more complicated. Let  $Q$  be a Q-node with subtrees  $T_1, \dots, T_n$ . Let  $x$  be a vertex contained only in maximal cliques of the subtree rooted at  $Q$ , and suppose that it is contained in maximal cliques of at least two subtrees. Then  $x$  is contained in every section  $s_i(Q)$  such that some maximal clique of  $T_i$  contains  $x$ . Figure 8b depicts the sections for the example in Fig. 7. It follows [18] that every vertex  $x$  is placed in the sections of exactly one node of  $T$ . For a Q-node,  $x$  is placed in consecutive sections of this node. Further, if  $x$  is placed in  $s_i(Q)$ , then  $x$  is contained in all cliques of  $T_i$ .

Let  $N$  be a node of the MPQ-tree. By  $G[N]$  we denote the subgraph induced by all the vertices in the sections of the subtree rooted at  $N$ . For a subtree  $T'$ , we denote the subgraph induced by the vertices in its sections by  $G[T']$ . Similarly, for a node  $N$ , let  $T[N]$  denote the subtree of  $T$  with the root  $N$ .

## 4 Characterizing Extendible Partial Representations

In this section, we restate the characterization of extendible partial representations by Klavík et al. [16] generalizing the theorem of Fulkerson and Gross [9].



**Fig. 9.** (a) Four maximal cliques with  $P(a) = \{u, v\}$ ,  $P(b) = \{y\}$ ,  $P(c) = \{x, y\}$ , and  $P(d) = \{y, z\}$  and the possible positions  $\downarrow_a, \downarrow_b, \downarrow_c$ , and  $\downarrow_d$  of their clique-points. (b) The corresponding open intervals  $I_a, I_b, I_c$  and  $I_d$ .

**The Interval Order  $\triangleleft$ .** Suppose that there exists a representation  $\mathcal{R}$  extending  $\mathcal{R}'$ . Then  $\mathcal{R}$  gives some ordering  $<$  of the maximal cliques from left to right. We want to show that pre-drawn intervals pose some constraints on this ordering.

For a maximal clique  $a$ , let  $P(a)$  denote the set of all pre-drawn intervals contained in  $a$ . Then  $P(a)$  restricts the possible position of the clique-point  $\text{cp}(a)$  to the points which are covered in  $\mathcal{R}'$  by the pre-drawn intervals of  $P(a)$  and no others. We denote this set of points by  $\downarrow_a$ ; see Fig. 9a. By [2], we get:  $\downarrow_a = (\bigcap_{u \in P(a)} \langle u \rangle') \setminus (\bigcup_{v \notin P(a)} \langle v \rangle')$ . We set  $\lrcorner(a) = \inf \downarrow_a$  and  $\rceil(a) = \sup \downarrow_a$ . We use an open interval  $I_a = (\lrcorner(a), \rceil(a))$  to represent  $\downarrow_a$ ; see Fig. 9b. There might be points between  $\lrcorner(a)$  and  $\rceil(a)$  where  $\text{cp}(a)$  cannot be placed.

For maximal cliques  $a$  and  $b$ , we write  $a \triangleleft b$  if  $\lrcorner(a) \leq \lrcorner(b)$ , i.e., if  $I_a$  is on the left of  $I_b$ . The definition of  $\triangleleft$  is quite natural, since  $a \triangleleft b$  implies that every extending representation  $\mathcal{R}$  has to place  $\text{cp}(a)$  to the left of  $\text{cp}(b)$ . The ordering  $\triangleleft$  is a so called *interval order* represented by open intervals, since  $a \triangleleft b$  if and only if the two intervals  $I_a$  and  $I_b$  are disjoint and  $I_a$  is on the left of  $I_b$ .

**Lemma 4.1 (Klavík et al. [16]).** *A partial representation  $\mathcal{R}'$  is extendible if and only if a consecutive ordering of the maximal cliques extending  $\triangleleft$  exists.*

In this paper, we extend these results by showing an additional property of  $\triangleleft$ . We say that a pair of intervals  $I_a$  and  $I_b$  *single overlaps* if both  $I_a \setminus I_b$  and  $I_b \setminus I_a$  are non-empty. If no single overlaps are allowed, every pair of intervals is either disjoint, or one interval is contained in the other.

**Lemma 4.2.** *No pair of intervals  $I_a$  and  $I_b$  single overlaps.*

**Lemma 4.3.** *If  $I_a \cap I_b = \emptyset$ , then at least one of  $P(a) \setminus P(b)$  and  $P(b) \setminus P(a)$  is non-empty. If  $I_a \subseteq I_b$ , then  $P(a) \supseteq P(b)$ .*

## 5 Locating Minimal Obstructions

Using the characterization of Section 4, we prove that every non-extendible partial representation contains one of the minimal obstructions defined in Section 2. Due to space limitations, we just explain the general strategy.

**Testing Extendibility.** For any two disjoint subtrees  $T_i$  and  $T_j$  of the PQ-tree  $T$ , we write  $T_i \triangleleft T_j$  if and only if there exist cliques  $a \in T_i$  and  $b \in T_j$  such that  $a \triangleleft b$ . For a given interval graph  $G$ , a PQ-tree  $T$  represents all feasible orderings of the maximal cliques. By Lemma 4.1, a partial representation is extendible if

and only if there exists a reordering  $T'$  of  $T$  such that the frontier of  $T'$  extends  $\triangleleft$ . The paper [16] gives the following algorithm for testing this.

The algorithm processes the PQ-tree  $T$  from the bottom to the root. When a P-node is processed, we test whether there exists a linear extension of  $\triangleleft$  on its subtrees. It exists if and only if  $\triangleleft$  induced on its subtrees is acyclic. If there exists a cycle, the PQ-tree cannot be reordered according to  $\triangleleft$ . When a Q-node is processed, there are two possible orderings of its subtrees, and we just check whether one of them is compatible with  $\triangleleft$ .

If a partial representation is not extendible, we know that this reordering fails in some node of  $T$ . A node which cannot be reordered is called *obstructed*. This insight is key in our strategy, since we can divide the argument according to the types of obstructed nodes of  $T$ . A set of maximal cliques *creates* an obstruction if the ordering of this set in  $\triangleleft$  already makes the node obstructed. We can show that every obstruction is created by at most three maximal cliques.

**5.1 Obstructed Leaves.** Suppose that some clique-point  $a$  cannot be placed at all, so  $\downarrow_a = \emptyset$ . Then  $\curvearrowright(a) = \infty$  and  $\curvearrowleft(a) = -\infty$ . In terms of  $\triangleleft$ , we get  $a \triangleleft a$ . By the definition of  $\downarrow_a$ , we derive a 1-BI obstruction; recall Fig. 6 (top-left).

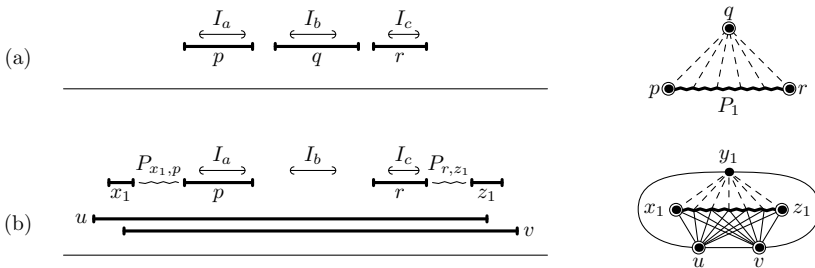
**5.2 Obstructed P-nodes.** Suppose that the reordering algorithm fails for a P-node. Then we have some cycle  $T'_1 \triangleleft T'_2 \triangleleft \dots \triangleleft T'_n \triangleleft T'_1$  on its subtrees. We first show that there exists a two-cycle  $T_1 \triangleleft T_2 \triangleleft T_1$ . This two-cycle is created by at most four maximal cliques, and we show that three are sufficient.

**Lemma 5.1 (The P-node case).** *If a P-node is obstructed, then  $G$  and  $\mathcal{R}$  contain an SE, a 1-FAT or a 1-BI obstruction.*

*Proof (Sketch).* For two maximal cliques  $a$  and  $b$  such that  $a \triangleleft b \triangleleft a$ , we get  $\curvearrowright(a) = \curvearrowleft(b) = \curvearrowleft(b) = \curvearrowright(a)$ , which leads to an SE obstruction; recall Fig. 4.

Assume that three maximal cliques create this obstruction. Let  $a, c \in T_1$  and  $b \in T_2$  such that  $a \triangleleft b \triangleleft c$ . Thus we have three non-intersecting intervals  $I_a, I_b$  and  $I_c$ ; see Fig. 10. We show that there always exist  $p \in P(a) \setminus P(b)$  and  $r \in P(c) \setminus P(a)$  which have to be pre-drawn like in Fig. 10.

If there also exists  $q \in P(b) \setminus P(a)$ , we get a 1-FAT obstruction for  $x_1 = p, y_1 = q$  and  $z_1 = r$ ; see Fig. 10a. If not, we use that  $I_b$  is in the middle. There



**Fig. 10.** (a) If  $P(b) \setminus P(a) \neq \emptyset$ , we get a 1-FAT obstruction. (b) If  $P(b) \setminus P(a) = \emptyset$ , we get a 1-BI obstruction.

are two intervals  $\langle u \rangle'$  and  $\langle v \rangle'$  such that each point of  $\langle u \rangle' \cap \langle v \rangle'$  outside  $I_b$  is covered by some pre-drawn interval not in  $P(b)$ . We choose  $x_1 \notin P(b)$  which covers  $\ell(v)$  and  $z_1 \notin P(b)$  which covers  $r(u)$ ; see Fig. 10. We suitably choose  $y_1 \in b$  non-adjacent to both  $x_1$  and  $z_1$ , and we get a 1-BI obstruction; recall Fig. 6 (top-left).  $\square$

**5.3 Obstructed Q-nodes.** An obstructed Q-node with subtrees  $T_1, \dots, T_n$  leads to a variety of minimal obstructions. First, we show that there are at most three maximal cliques creating this obstruction, which is harder than for P-nodes. These cliques are contained in two or three different subtrees.

**Lemma 5.2 (The Q-node case A).** *In the case of two different subtrees,  $G$  and  $\mathcal{R}'$  contain an SE, a 1-FAT, a 2-FAT, a 1-BI, or a 2-BI obstruction.*

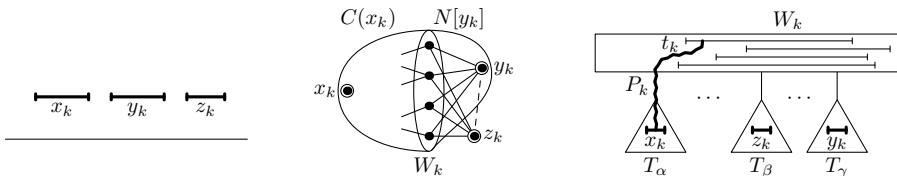
**Lemma 5.3 (The Q-node case B).** *In the case of three different subtrees,  $G$  and  $\mathcal{R}'$  contain a  $k$ -FAT,  $k$ -BI,  $k$ -FS,  $k$ -EFS,  $k$ -FB,  $k$ -FDS,  $k$ -EFDS,  $k$ -FNS, or a  $(k, \ell)$ -CE obstruction.*

The proof of Lemma 5.3 deals with many situations which lead to different obstructions. We assume that  $a \in T_\alpha$ ,  $b \in T_\beta$  and  $c \in T_\gamma$  such that  $\alpha < \beta < \gamma$  and  $a \triangleleft b \triangleright c$ . We know that both  $I_a$  and  $I_c$  appear on the left of  $I_b$ , but they might be in inclusion or disjoint. For instance, if  $I_a$  is on the left of  $I_c$ , we would like to get  $p \in P(a)$ ,  $q \in P(c)$  and  $r \in P(b)$ , similarly as in Fig. 10a. We get different cases according to which of these exist and how they overlap.

The following lemma is repeatedly used in the proof. It says that if pre-drawn intervals are ordered differently than in the Q-node, we find a  $k$ -FAT obstruction.

**Lemma 5.4 ( $k$ -FAT).** *Let  $a, b$  and  $c$  be three cliques of  $T[Q]$  contained respectively in  $T_\alpha, T_\beta$  and  $T_\gamma$  for  $\alpha < \beta < \gamma$ . Let  $x_k \in P(a)$ ,  $y_k \in P(c)$  and  $z_k \in P(b)$  be three disjoint pre-drawn intervals such that  $\langle y_k \rangle'$  is between  $\langle x_k \rangle'$  and  $\langle z_k \rangle'$ . Then  $G[Q]$  and  $\mathcal{R}'$  contain a  $k$ -FAT obstruction.*

*Proof (Sketch).* See Fig. 11. We give an inductive argument. Either we find a 1-FAT or a 2-FAT obstruction directly, or we recurse on a smaller part of the Q-node where we find an almost  $(k - 1)$ -FAT obstruction, in which  $x_{k-1}$  is free. Together with  $x_k$  and some other vertices, we construct a  $k$ -FAT obstruction.



**Fig. 11.** On the left, the position of the pre-drawn intervals  $x_k, y_k$  and  $z_k$ . In the middle, the construction of  $W_k \subseteq N[y_k]$  in  $G[Q]$ . On the right, the Q-node with the three considered subtrees and the intervals of  $W_k$  depicted in their sections.



Suppose that there exists a path from  $x_k$  to  $z_k$  such that all inner vertices are non-adjacent to  $y_k$ . Then we get a 1-FAT obstruction. Otherwise, let  $C(x_k)$  be the connected component of  $G[Q] \setminus N[y_k]$  containing  $x_k$ ; by our assumption,  $z_k \notin C(x_k)$ . We denote by  $W_k$  the subset of the vertices of  $N[y_k]$  adjacent to some vertex of  $C(x_k)$ ; see Fig. 11 (middle). It can be shown that every vertex of  $W_k$  is also adjacent to  $z_k$ . Therefore,  $W_k \subseteq s_\beta(Q) \cap s_\gamma(Q)$ ; see Fig. 11 (right).

Let  $t_k$  be a vertex of  $W_k$  which ends in the sections of the Q-node most to the left;  $t_k$  is the top interval in Fig. 11. Let  $P_k$  be a shortest path from  $x_k$  to  $t_k$ , with all inner vertices in  $C(x_k)$ . We denote the component of  $G[Q] \setminus W_k$  containing  $y_k$  by  $C(y_k)$  and the one containing  $z_k$  by  $C(z_k)$ . We put  $y_{k-1} = z_k$  and  $z_{k-1} = y_k$ . It is not possible that  $t_k$  is adjacent to every vertex in  $C(y_k)$ , otherwise the MPQ-tree would be incorrect. Therefore, we choose a non-neighbor  $x_{k-1} \in C(y_k)$ . If  $C(y_k) \neq C(z_k)$ , together with some path  $P_{k-1}$  from  $x_{k-1}$  to  $z_{k-1}$ , we get a 2-FAT obstruction, so  $k = 2$ . But if  $C(y_k) = C(z_k)$ , we cannot guarantee that the inner vertices of this path are non-adjacent to  $y_{k-1}$ . We solve this issue by applying the entire argument of this proof recursively to  $C(y_k)$ .

We know that  $t_k$  stretches from  $C(x_k)$  to  $z_k$ , completely covering  $y_k$ . So  $x_{k-1}$  has to be placed to the right of  $z_k = y_{k-1}$  in every extending representation. We assume that  $x_{k-1}$  is pre-drawn on the right of  $y_{k-1}$  and repeat the same argument for the graph induced by  $C(y_k)$ , where the role of  $x_k$ ,  $y_k$  and  $z_k$  is played by  $x_{k-1}$ ,  $y_{k-1}$  and  $z_{k-1}$ , respectively. By the induction hypothesis, we find a  $(k-1)$ -FAT obstruction. By making  $x_{k-1}$  free and adding  $x_k$ ,  $t_k$  and  $P_k$ , we get a  $k$ -FAT obstruction in the original partial representation.  $\square$

This lemma is used to locate other obstructions. Suppose that we have  $x_k$ ,  $z_k$ ,  $u$  and  $v$  pre-drawn as in Fig. 10b. If we find  $y_k$  located suitably in the Q-node, then we can assume  $\langle y_k \rangle'$  is pre-drawn between  $\langle x_k \rangle'$  and  $\langle z_k \rangle'$ . Therefore, we apply Lemma 5.4 for a different partial representation and we get a  $k$ -FAT obstruction. With free  $y_k$  and together with  $u$  and  $v$ , it gives a  $k$ -BI obstruction.

#### 5.4 Proof of the Main Theorem. We just put all results together:

*Proof (Theorem 1.1).* If  $G$  and  $\mathcal{R}'$  contain one of the obstructions, then  $\mathcal{R}'$  is non-extendible by Lemma 2.1. It remains to prove the converse. If  $G$  is not an interval graph, it contains according to [19] an LB obstruction. Otherwise,  $G$  is an interval graph and there exists an MPQ-tree  $T$  for it. By Lemma 4.1, we know that a partial representation  $\mathcal{R}'$  is extendible if and only if  $T$  can be reordered according to  $\triangleleft$ . If it cannot be reordered, the reordering algorithm fails in some node of  $T$ . If this reordering fails in a leaf, we get a 1-BI obstruction. If it fails in a P-node, we get a 1-FAT or a 1-CI obstruction by Lemma 5.1. And if it fails in a Q-node, we get one of the minimal obstructions by Lemmas 5.2 and 5.3.  $\square$

Since every minimal obstruction contains at most four pre-drawn intervals, a partial representation  $\mathcal{R}'$  is extendible if and only if every quadruple of pre-drawn intervals is extendible by itself (Corollary 1.2). Since minimal obstructions are build constructively, we get as Corollary 1.3 a certifying algorithm for the partial representation extension problem.

## 6 Open Problems

*Circle graphs* (CIRCLE) are intersection graphs of chords of a circle. *Function graphs* (FUN) are intersection graphs of continuous functions  $f : [0, 1] \rightarrow \mathbb{R}$  and *permutation graphs* (PERM) are function graphs which can be represented by linear functions. *Proper interval graphs* (PROPER INT) are intersection graphs of intervals in which no interval is a proper subset of another interval. *Unit interval graphs* (UNIT INT) are intersection graphs of intervals of length one.

*Problem 6.1.* What are minimal obstructions for extendible partial representations of the classes CIRCLE, FUN, PERM, PROPER INT, and UNIT INT?

The *bounded representation problems* generalize the partial representation problems [2, 14]. In 1984, Skrien [22] introduced the *chronological ordering problem* which even generalizes bounded representations.

*Problem 6.2.* What are minimal obstructions for the chronological ordering problem and for the bounded representation problem?

## References

1. Angelini, P., Battista, G.D., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. In: SODA 2010, pp. 202–221 (2010)
2. Balko, M., Klavík, P., Otachi, Y.: Bounded representations of interval and proper interval graphs. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) Algorithms and Computation. LNCS, vol. 8283, pp. 535–546. Springer, Heidelberg (2013)
3. Benzer, S.: On the topology of the genetic fine structure. Proc. Nat. Acad. Sci. U.S.A. **45**, 1607–1620 (1959)
4. Bläsius, T., Rutter, I.: Simultaneous PQ-ordering with applications to constrained embedding problems. In: SODA 2013, pp. 1030–1043 (2013)
5. Booth, K., Lueker, G.: Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. J. Comput. System Sci. **13**, 335–379 (1976)
6. Chaplick, S., Dorbec, P., Kratochvíl, J., Montassier, M., Stacho, J.: Contact representations of planar graph: Rebuilding is hard. In: WG 2014 (2014)
7. Chaplick, S., Fulek, R., Klavík, P.: Extending partial representations of circle graphs. In: Wismath, S., Wolff, A. (eds.) GD 2013. LNCS, vol. 8242, pp. 131–142. Springer, Heidelberg (2013)
8. Colbourn, C.J., Booth, K.S.: Linear times automorphism algorithms for trees, interval graphs, and planar graphs. SIAM J. Comput. **10**(1), 203–225 (1981)
9. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pac. J. Math. **15**, 835–855 (1965)
10. Hajós, G.: Über eine Art von Graphen. Internat. Math. News **11**, 65 (1957)
11. Jelínek, V., Kratochvíl, J., Rutter, I.: A kuratowski-type theorem for planarity of partially embedded graphs. Comput. Geom. **46**(4), 466–492 (2013)
12. Kendall, D.G.: Incidence matrices, interval graphs and seriation in archaeology. Pac. J. Math **28**(3), 565–570 (1969)
13. Klavík, P., Kratochvíl, J., Krawczyk, T., Walczak, B.: Extending partial representations of function graphs and permutation graphs. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 671–682. Springer, Heidelberg (2012)

14. Klavík, P., Kratochvíl, J., Otachi, Y., Rutter, I., Saitoh, T., Saumell, M., Vyskočil, T.: Extending partial representations of proper and unit interval graphs. In: Ravi, R., Gørtz, I.L. (eds.) SWAT 2014. LNCS, vol. 8503, pp. 253–264. Springer, Heidelberg (2014)
15. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T.: Extending partial representations of subclasses of chordal graphs. In: Chao, K.-M., Hsu, T., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 444–454. Springer, Heidelberg (2012)
16. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T., Vyskočil, T.: Extending partial representations of interval graphs. CoRR, abs/1306.2182 (2013)
17. Klavík, P., Kratochvíl, J., Vyskočil, T.: Extending partial representations of interval graphs. In: Ogihara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 276–285. Springer, Heidelberg (2011)
18. Korte, N., Möhring, R.: An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.* **18**(1), 68–81 (1989)
19. Lekkerkerker, C., Boland, D.: Representation of finite graphs by a set of intervals on the real line. *Fund. Math.* **51**, 45–64 (1962)
20. Patrignani, M.: On extending a partial straight-line drawing. *Int. J. Found. Comput. Sci.* **17**(5), 1061–1070 (2006)
21. Roberts, F.S.: *Discrete Mathematical Models, with Applications to Social, Biological, and Environmental Problems*. Prentice-Hall, Englewood Cliffs (1976)
22. Skrien, D.: Chronological orderings of interval graphs. *Discrete Appl. Math.* **8**(1), 69–83 (1984)

# Faster Algorithms for Computing the $R^*$ Consensus Tree

Jesper Jansson<sup>1</sup>(✉), Wing-Kin Sung<sup>2,3</sup>, Hoa Vu<sup>4</sup>, and Siu-Ming Yiu<sup>5</sup>

<sup>1</sup> Laboratory of Mathematical Bioinformatics, Institute for Chemical Research,  
Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan

`jj@kuicr.kyoto-u.ac.jp`

<sup>2</sup> School of Computing, National University of Singapore,  
13 Computing Drive, Singapore 117417, Singapore

`ksung@comp.nus.edu.sg`

<sup>3</sup> Genome Institute of Singapore, 60 Biopolis Street, Genome,  
Singapore 138672, Singapore

<sup>4</sup> Department of Computer Science and Engineering,  
University of Minnesota – Twin Cities, Minneapolis, MN, USA

`hoavu89@gmail.com`

<sup>5</sup> Department of Computer Science, The University of Hong Kong,  
Pokfulam Road, Hong Kong, China

`smyiu@cs.hku.hk`

**Abstract.** The fastest known algorithms for computing the  $R^*$  consensus tree of  $k$  rooted phylogenetic trees with  $n$  leaves each and identical leaf label sets run in  $O(n^2\sqrt{\log n})$  time when  $k = 2$  (ref. [10]) and  $O(kn^3)$  time when  $k \geq 3$  (ref. [4]). This paper shows how to compute it in  $O(n^2)$  time for  $k = 2$ ,  $O(n^2 \log^{4/3} n)$  time for  $k = 3$ , and  $O(n^2 \log^{k+2} n)$  time for unbounded  $k$ .

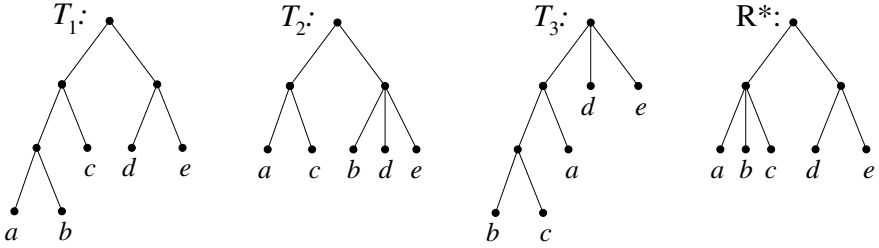
## 1 Introduction

Distinctly leaf-labeled, unordered trees known as *phylogenetic trees* are used by scientists to describe evolutionary history [8, 13]. Given a set  $\mathcal{S}$  of phylogenetic trees with the same leaf labels but different branching structures, a single phylogenetic tree that summarizes the trees in  $\mathcal{S}$  according to some well-defined rule is called a *consensus tree* [4, 8, 13]. Consensus trees are used when dealing with unreliable data; e.g., to infer an accurate phylogenetic tree for a fixed set of species, one may first construct a collection of alternative trees by applying resampling techniques such as bootstrapping to the same data set, by running different tree construction algorithms, or by using many independent data sets, and then compute a consensus tree from the obtained trees.

A number of different consensus trees have been defined and studied in the literature; see [4], Chapter 30 in [8], or Chapter 8.4 in [13] for some surveys. This paper deals with one particular consensus tree called the  $R^*$  consensus tree [4],

---

Jesper Jansson: Funded by The Hakubi Project and KAKENHI grant number 26330014.



**Fig. 1.** An example. Let  $S = \{T_1, T_2, T_3\}$  as above. Then  $\mathcal{R}_{maj} = \{ab|d, ab|e, ac|d, ac|e, de|a, bc|d, bc|e, de|b, de|c\}$  and the  $R^*$  consensus tree of  $S$  is the tree on the right.

defined in Section 1.1 below. The  $R^*$  consensus tree has several nice mathematical properties [7]. On the negative side, the existing algorithms for building it [4, 10, 11] are rather slow. To alleviate this issue, we present faster algorithms.

**1.1 Definitions and Notation**

In this paper, a *phylogenetic tree* is a rooted, unordered, leaf-labeled tree in which every internal node has at least two children and all leaves have different labels. See Fig. 1 for some examples. (*Unrooted* phylogenetic trees are also useful in many contexts [8], but will not be considered here.) Phylogenetic trees are called “trees” from here on, and every leaf in a tree is identified with its label.

Let  $T$  be a tree. The set of all nodes in  $T$  and the set of all leaves in  $T$  are denoted by  $V(T)$  and  $\Lambda(T)$ , respectively. For any  $u \in V(T)$ ,  $T^u$  is the subtree of  $T$  rooted at  $u$ . For any  $X \subseteq V(T)$ ,  $lca^T(X)$  is the lowest common ancestor in  $T$  of the nodes in  $X$ ; when  $|X| = 2$ , we simplify the notation to  $lca^T(u, v)$ , where  $X = \{u, v\}$ , and if  $T$  is unambiguous, we sometimes just write  $lca(u, v)$ .

A *triplet* is a tree with exactly three leaves. Suppose  $t$  is a triplet with  $\Lambda(t) = \{x, y, z\}$ . If  $t$  is non-binary, it has one internal node; in this case,  $t$  is called a *fan triplet* and is denoted by  $x|y|z$ . Otherwise,  $t$  is binary and has two internal nodes; in this case,  $t$  is called a *resolved triplet* and is denoted by  $xy|z$  where  $lca^t(x, y)$  is a proper descendant of  $lca^t(x, z) = lca^t(y, z)$ . Thus, there are four possible triplets  $x|y|z, xy|z, xz|y, yz|x$  for any set of three leaves  $\{x, y, z\}$ . For any tree  $T$  and  $\{x, y, z\} \subseteq \Lambda(T)$ ,  $x|y|z$  is said to be *consistent with  $T$*  if  $lca^T(x, y) = lca^T(x, z) = lca^T(y, z)$ , and  $xy|z$  is *consistent with  $T$*  if  $lca^T(x, y)$  is a proper descendant of  $lca^T(x, z) = lca^T(y, z)$ . Let  $T|_{\{x,y,z\}}$  be the unique triplet with leaf set  $\{x, y, z\}$  that is consistent with  $T$ . For any tree  $T$ , let  $r(T)$  be the set of resolved triplets consistent with  $T$  and let  $t(T)$  be the set of *all* triplets (resolved triplets as well as fan triplets) consistent with  $T$ , i.e., define  $r(T) = \{T|_{\{x,y,z\}} : \{x, y, z\} \subseteq \Lambda(T) \text{ and } T|_{\{x,y,z\}} \text{ is a resolved triplet}\}$  and  $t(T) = \{T|_{\{x,y,z\}} : \{x, y, z\} \subseteq \Lambda(T)\}$ .

Next, let  $\mathcal{S} = \{T_1, \dots, T_k\}$  be a given set of trees with  $\Lambda(T_1) = \dots = \Lambda(T_k) = L$ . Write  $n = |L|$ . For any  $\{a, b, c\} \subseteq L$ , define  $\#ab|c$  as the number of trees  $T_i \in \mathcal{S}$  for which  $ab|c \in t(T_i)$ . The *set of majority resolved triplets*, denoted

by  $\mathcal{R}_{maj}$ , is defined as  $\{abc : a, b, c \in L \text{ and } \#ab|c > \max\{\#ac|b, \#bc|a\}\}$ . (Note that the fan triplets consistent with the trees in  $\mathcal{S}$  have no impact here.) An  $R^*$  consensus tree of  $\mathcal{S}$  is a tree  $\tau$  with  $\Lambda(\tau) = L$  that satisfies  $r(\tau) \subseteq \mathcal{R}_{maj}$  and that maximizes the number of internal nodes. See Fig. 1 for an example.

For any leaf label set  $L$ , a cluster of  $L$  is any nonempty subset of  $L$ , and a tree  $T$  is said to include a cluster  $A$  of  $L$  if  $T$  contains a node  $u$  such that  $\Lambda(T^u) = A$ . Let  $\mathcal{R}$  be a set of triplets over a leaf label set  $L = \bigcup_{r \in \mathcal{R}} \Lambda(r)$  such that for each  $\{x, y, z\} \subseteq L$ , at most one of  $x|y|z$ ,  $xy|z$ ,  $xz|y$ , and  $yz|x$  belongs to  $\mathcal{R}$ . A cluster  $A$  of  $L$  is called a strong cluster of  $\mathcal{R}$  if  $aa'|x \in \mathcal{R}$  for all  $a, a' \in A$  with  $a \neq a'$  and all  $x \in L \setminus A$ . Furthermore,  $L$  as well as every singleton set of  $L$  is also defined to be a strong cluster of  $\mathcal{R}$ . Strong clusters provide an alternative characterization of  $R^*$  consensus trees, stated in the last part of the next lemma:

**Lemma 1.** [4, 10] *The  $R^*$  consensus tree always exists, is unique, and includes every strong cluster of  $\mathcal{R}_{maj}$  and no other clusters.*

### 1.2 Previous Work

The  $R^*$  consensus tree can be computed in  $O(kn^3)$  time, where  $k = |\mathcal{S}|$  and  $n = |L|$ , by an algorithm from [4]: First construct  $r(T_i)$  for all  $T_i \in \mathcal{S}$  in  $O(kn^3)$  time, then construct  $\mathcal{R}_{maj}$  by counting the occurrences in the  $r(T_i)$ -sets of the different resolved triplets for every  $\{x, y, z\} \in L$  in  $O(kn^3)$  total time, and finally apply the  $O(n^3)$ -time strong cluster algorithm from Corollary 2.2 in [5] to  $\mathcal{R}_{maj}$ . For  $k = 2$ , an older algorithm for computing the so-called RV-III tree of two input trees in  $O(n^3)$  time [11] can also be used [4] to achieve the same running time.

Since  $\mathcal{R}_{maj}$  may contain  $\Omega(n^3)$  elements, any method that explicitly constructs  $\mathcal{R}_{maj}$  requires  $\Omega(n^3)$  time. For the special case of  $k = 2$ , it was shown in [10] that the  $R^*$  consensus tree can in fact be computed in  $O(n^2\sqrt{\log n})$  ( $= o(n^3)$ ) time. The algorithm from [10] is reviewed in Section 1.3.

### 1.3 Overview and Organization of the Paper

To compute the  $R^*$  consensus tree without constructing  $\mathcal{R}_{maj}$ , the algorithm in [10] for  $k = 2$  and the new algorithms in this paper follow the same basic strategy, summarized as Algorithm `R*_consensus_tree` in Fig. 2. To explain the details, some additional definitions are needed.

Suppose that  $\mathcal{R}$  is a given set of triplets over a leaf label set  $L = \bigcup_{r \in \mathcal{R}} \Lambda(r)$  such that for each  $\{x, y, z\} \subseteq L$ , at most one of  $x|y|z$ ,  $xy|z$ ,  $xz|y$ , and  $yz|x$  belongs to  $\mathcal{R}$ . For each  $a, b \in L$  with  $a \neq b$ , define  $s_{\mathcal{R}}(a, b) = |\{y \in L : ab|y \in \mathcal{R}\}|$ , and for each  $a \in L$ , define  $s_{\mathcal{R}}(a, a) = |L| - 1$ . A cluster  $A$  of  $L$  is called an Apresjan cluster of  $s_{\mathcal{R}}$  if  $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$  for all  $a, a' \in A$  and  $x \in L \setminus A$ . Since every strong cluster of  $\mathcal{R}$  is an Apresjan cluster of  $s_{\mathcal{R}}$  [4, 10], we see that in the case  $\mathcal{R} = \mathcal{R}_{maj}$ , the set of Apresjan clusters of  $s_{\mathcal{R}_{maj}}$  forms a superset of the set of strong clusters of  $\mathcal{R}_{maj}$ . Moreover, by Theorem 2.3 in [5], there are

**Algorithm  $R^*$ \_consensus\_tree**  
**Input:** A set  $\mathcal{S} = \{T_1, \dots, T_k\}$  of trees with  $\Lambda(T_1) = \dots = \Lambda(T_k) = L$   
**Output:** The  $R^*$  consensus tree of  $\mathcal{S}$

- 1: Compute and store  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $a, b \in L$
- 2: Compute the Apresjan clusters of  $s_{\mathcal{R}_{maj}}$
- 3: **for** each Apresjan cluster  $A$  of  $s_{\mathcal{R}_{maj}}$  **do**
- 4: Determine if  $A$  is a strong cluster of  $\mathcal{R}_{maj}$
- 5: **end for**
- 6: Let  $C$  be the set of strong clusters of  $\mathcal{R}_{maj}$ , and build a tree  $T$  which includes all clusters in  $C$  and no other clusters of  $L$
- 7: Output  $T$

**Fig. 2.** Algorithm  $R^*$ \_consensus\_tree

$O(n)$  Apresjan clusters of  $s_{\mathcal{R}_{maj}}$  and they form a nested hierarchy on  $L$ , i.e., a tree, which can be constructed in  $O(n^2)$  time with the method of Corollary 2.1 in [5] when the value of  $s_{\mathcal{R}_{maj}}(a, b)$  for any  $a, b \in L$  is available in  $O(1)$  time.

Now, the idea behind Algorithm  $R^*$ \_consensus\_tree is to first compute a superset of the set of strong clusters of  $\mathcal{R}_{maj}$ , namely the Apresjan clusters of  $s_{\mathcal{R}_{maj}}$  (Steps 1 and 2), then remove any clusters that are not strong clusters of  $\mathcal{R}_{maj}$  (Steps 3–5), and return a tree that includes precisely the remaining clusters (Steps 6–7). By Lemma 1, this tree is the  $R^*$  consensus tree.

The algorithm’s time complexity depends on various factors. As shown in [10], if  $k = 2$  then computing the values of  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $a, b \in L$  in Step 1 can be done in  $O(n^2 \sqrt{\log n})$  time in total, while all other steps take  $O(n^2)$  time. Section 2 below improves it to  $O(n^2)$ , yielding an  $O(n^2)$ -time solution for  $k = 2$ .

For  $k \geq 3$ , we observe that Steps 2, 6, and 7 do not depend on  $k$ , so these steps take a total of  $O(n^2)$  time as in [10]. However, Steps 1 and 3–5 have to be modified; for example, the condition from Lemma 13 in [10] for checking if a given cluster is a strong cluster of  $\mathcal{R}_{maj}$  does not work if  $k = 3$ . As for Step 1, Sections 3.1–3.3 show how to compute  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $a, b \in L$  in  $O(n^2 \log^{4/3} n)$  time when  $k = 3$ , and Section 4.1 in  $O(n^2 \log^k n)$  time for unbounded  $k$ . For Steps 3–5, Section 3.4 gives an  $O(n^2 \alpha(n))$ -time solution when  $k = 3$ , where  $\alpha(n)$  is the inverse Ackermann function of  $n$ , while Section 4.2 gives an  $O(n^2 \log^{k+2} n)$ -time solution for unbounded  $k$ . In summary, we obtain:

**Theorem 1.** *Let  $\mathcal{S}$  be an input set of  $k$  trees with  $n$  leaves each and identical leaf label sets. The  $R^*$  consensus tree of  $\mathcal{S}$  can be computed in:*

- $O(n^2)$  time when  $k = 2$ ;
- $O(n^2 \log^{4/3} n)$  time when  $k = 3$ ; and
- $O(n^2 \log^{k+2} n)$  time when  $k$  is unbounded.

Thus, if  $k < \frac{\log n}{(\log \log n)^{1+\epsilon}}$  for some  $\epsilon > 0$ , the time complexity is subcubic in  $n$ .

Due to space constraints, most of the proofs have been omitted from the conference proceedings version of this paper.

## 2 Computing the R\* Consensus Tree When $k = 2$

This section proves that  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $a, b \in L$  with  $a \neq b$  can be computed in  $O(n^2)$  time in total when  $k = 2$ , thereby reducing the time complexity of Step 1 of Algorithm `R*_consensus_tree` in Section 1.3 (and hence the algorithm’s overall running time) to  $O(n^2)$ .

Recall that  $s_{\mathcal{R}_{maj}}(a, b) = |\{w : ab|w \in \mathcal{R}_{maj}\}|$  for any  $a, b \in L$  with  $a \neq b$ , and  $s_{\mathcal{R}_{maj}}(a, a) = |L| - 1$  for any  $a \in L$ . By definition,  $ab|w \in \mathcal{R}_{maj}$  if and only if it is consistent with both  $T_1$  and  $T_2$ , or it is consistent with one of  $T_1$  and  $T_2$  and  $a|b|w$  is consistent with the other tree. By Corollary 1 in [10],  $s_{\mathcal{R}_{maj}}(a, b) = \text{count}_{r,r}(a, b) + \text{count}_{r,f}(a, b) + \text{count}_{f,r}(a, b)$  for every  $a, b \in L$  with  $a \neq b$ , where  $\text{count}_{r,r}(a, b) = |\{w \in L \setminus \{a, b\} : ab|w \in t(T_1) \cap t(T_2)\}|$ ,  $\text{count}_{r,f}(a, b) = |\{w \in L \setminus \{a, b\} : ab|w \in t(T_1), a|b|w \in t(T_2)\}|$ , and  $\text{count}_{f,r}(a, b) = |\{w \in L \setminus \{a, b\} : a|b|w \in t(T_1), ab|w \in t(T_2)\}|$ . It was shown in [10] that  $\text{count}_{r,r}(a, b)$ ,  $\text{count}_{r,f}(a, b)$ , and  $\text{count}_{f,r}(a, b)$  for all  $a, b \in L$  can be calculated in  $O(n^2\sqrt{\log n})$ ,  $O(n^2)$ , and  $O(n^2)$  total time, respectively. We now eliminate the bottleneck.

**Lemma 2.** *For every  $a, b \in L$ , it holds that  $\text{count}_{r,r}(a, b) = |L| - |\Lambda(T_1^{lca(a,b)})| - |\Lambda(T_2^{lca(a,b)})| + |\Lambda(T_1^{lca(a,b)} \cap \Lambda(T_2^{lca(a,b)})|$ .*

**Lemma 3.**  *$\text{count}_{r,r}(a, b)$  for all  $a, b \in L$  can be computed in  $O(n^2)$  time in total.*

*Proof.* For  $i \in \{1, 2\}$ , compute and store all values of  $|\Lambda(T_i^u)|$ , where  $u \in V(T_i)$ , in  $O(n)$  time by doing a bottom-up traversal of each tree. Also, compute and store all values of  $|\Lambda(T_1^u) \cap \Lambda(T_2^v)|$ , where  $u \in V(T_1)$  and  $v \in V(T_2)$ , in  $O(n^2)$  time by the postorder traversal-based method used in Lemma 7.1 in [1]. Preprocess  $T_1$  and  $T_2$  in  $O(n)$  time so that any subsequent *lca*-query can be answered in  $O(1)$  time [2,9]. Next, for each  $a, b \in L$ , obtain  $\text{count}_{r,r}(a, b)$  in  $O(1)$  time by applying the formula in Lemma 2. The total running time is  $O(n^2)$ .  $\square$

## 3 Computing the R\* Consensus Tree When $k = 3$

We now focus on the case  $k = 3$ . Sections 3.1–3.3 and Section 3.4 describe how to implement Step 1 and Steps 3–5, respectively, of Algorithm `R*_consensus_tree`.

### 3.1 Computing $s_{\mathcal{R}_{maj}}$ When $k = 3$

Suppose  $\mathcal{S} = \{T_1, T_2, T_3\}$ . For every  $ab|w \in \mathcal{R}_{maj}$ , there are three possibilities:

**Lemma 4.** *For any  $a, b, w \in L$ ,  $ab|w \in \mathcal{R}_{maj}$  if and only if either*

1.  $ab|w$  is consistent with  $T_1, T_2$ , and  $T_3$ ; or
2.  $ab|w$  is consistent with  $T_i$  and  $T_j$  but not  $T_k$  for  $\{i, j, k\} = \{1, 2, 3\}$ ; or
3.  $ab|w$  is consistent with one of  $T_1, T_2, T_3$ , and  $a|b|w$  with the other two.



To count the triplets covered by the different cases in Lemma 4, define:

$$\begin{cases} \text{count}_{r,r,r}(a,b) = |\{w \in L \setminus \{a,b\} : ab|w \in t(T_1) \cap t(T_2) \cap t(T_3)\}| \\ \text{count}_{r,r,*}^{T_i,T_j}(a,b) = |\{w \in L \setminus \{a,b\} : ab|w \in t(T_i) \cap t(T_j)\}|, i,j \in \{1,2,3\}, i < j \\ \text{count}_{r,f,f}^{T_i}(a,b) = |\{w \in L \setminus \{a,b\} : ab|w \in t(T_i) \text{ and } a|b|w \text{ is consistent with} \\ \text{the other two trees}\}|, \text{ for } i \in \{1,2,3\} \end{cases}$$

Then,  $s_{\mathcal{R}_{maj}}(a,b)$  can be expressed as in the next lemma.

**Lemma 5.** *Let  $a,b \in L$  with  $a \neq b$ . Then  $s_{\mathcal{R}_{maj}}(a,b) = \sum_{i=1}^3 \text{count}_{r,f,f}^{T_i}(a,b) + \sum_{1 \leq i < j \leq 3} \text{count}_{r,r,*}^{T_i,T_j}(a,b) - 2\text{count}_{r,r,r}(a,b)$ .*

For each pair  $i,j \in \{1,2,3\}$  with  $i < j$ , the values of  $\text{count}_{r,r,*}^{T_i,T_j}(a,b)$  for all  $a,b \in L$  can be obtained in  $O(n^2)$  time by the method from Lemma 3 in Section 2 with  $T_i$  and  $T_j$  as the two input trees. The next subsections show how to calculate the values of  $\text{count}_{r,r,r}(a,b)$  for all  $a,b \in L$  in  $O(n^2 \log^{4/3} n)$  time (Lemma 9 in Section 3.2) and  $\text{count}_{r,f,f}^{T_i}(a,b)$  for all  $a,b \in L$  for each  $i \in \{1,2,3\}$  in  $O(n^2)$  time (Lemma 12 in Section 3.3). Then, we can apply the formula in Lemma 5 to get each value of  $s_{\mathcal{R}_{maj}}(a,b)$  in  $O(1)$  time. In summary:

**Lemma 6.** *When  $k = 3$ , the values of  $s_{\mathcal{R}_{maj}}(a,b)$  for all  $a,b \in L$  can be computed in  $O(n^2 \log^{4/3} n)$  time in total.*

### 3.2 Computing $\text{count}_{r,r,r}$

First, rewrite  $\text{count}_{r,r,r}(a,b)$  in a way analogous to the expression in Lemma 2:

**Lemma 7.** *For every  $a,b \in L$ ,  $\text{count}_{r,r,r}(a,b) = |L| - \sum_{i=1}^3 |\Lambda(T_i^{lca(a,b)})| + \sum_{1 \leq i < j \leq 3} |\Lambda(T_i^{lca(a,b)}) \cap \Lambda(T_j^{lca(a,b)})| - |\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$ .*

**Lemma 8.** *Let  $a \in L$  be fixed. Then the values of  $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$  for all  $b \in L \setminus \{a\}$  can be computed in  $O(n \log^{4/3} n)$  time in total.*

*Proof.* For  $w \in L \setminus \{a\}$  and  $i \in \{1,2,3\}$ , let  $d^{T_i}(w)$  be the distance in  $T_i$  from  $a$  to  $lca(a,w)$ . For any  $b,w \in L \setminus \{a\}$  and  $i \in \{1,2,3\}$ ,  $w \in \Lambda(T_i^{lca(a,b)})$  if and only if  $d^{T_i}(w) \leq d^{T_i}(b)$ . Thus, for  $b \in L \setminus \{a\}$ ,  $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})| = |\{w \in L \setminus \{a,b\} : d^{T_1}(w) \leq d^{T_1}(b), d^{T_2}(w) \leq d^{T_2}(b), d^{T_3}(w) \leq d^{T_3}(b)\}|$ .

Represent each  $w \in L \setminus \{a\}$  as a 3D point with coordinates  $(d^{T_1}(w), d^{T_2}(w), d^{T_3}(w))$ . For any  $b \in L \setminus \{a\}$ ,  $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$  equals the number of points on or inside the box  $[1 : d^{T_1}(b)] \times [1 : d^{T_2}(b)] \times [1 : d^{T_3}(b)]$ . Use Corollary 4.1 in [6] for offline orthogonal range counting in 3D to obtain these numbers for all  $b \in L \setminus \{a\}$  in  $O(n \log^{3-2+1/3} n) = O(n \log^{4/3} n)$  total time.  $\square$

**Lemma 9.** *The values of  $\text{count}_{r,r,r}(a,b)$  for all  $a,b \in L$  can be computed in  $O(n^2 \log^{4/3} n)$  total time.*

### 3.3 Computing $\text{count}_{r,f,f}^{T_i}$

This subsection describes how to compute all values of  $\text{count}_{r,f,f}^{T_1}(a, b) = |\{w \in L \setminus \{a, b\} : ab|w \in t(T_1), a|b|w \in t(T_2), \text{ and } a|b|w \in t(T_3)\}|$ , where  $a, b \in L$ . (The two functions  $\text{count}_{r,f,f}^{T_2}$  and  $\text{count}_{r,f,f}^{T_3}$  can be computed in the same way.)

Suppose that  $a \in L$  is fixed. Let  $v_0 = a, v_1, \dots, v_p$  be the path in  $T_3$  from leaf  $a$  to the root of  $T_3$ . For  $j \in \{1, \dots, p\}$ , define  $W_j = \Lambda(T_3^{v_j}) \setminus \Lambda(T_3^{v_{j-1}})$ . Importantly,  $\{W_1, \dots, W_p\}$  forms a partition of  $L \setminus \{a\}$ . For any  $S \subseteq L$  and  $b \in S$ , define  $\sigma^{T_1, -T_2}(S, b) = |\{w \in S : ab|w \in t(T_1) \text{ and } a|b|w \in t(T_2)\}|$ . Lemma 10 explains how to use  $\sigma^{T_1, -T_2}(S, b)$  to compute  $\text{count}_{r,f,f}^{T_1}(a, b)$ .

**Lemma 10.** *For any  $W_j, j \in \{1, \dots, p\}$ , and any  $b \in W_j$ , let  $c_b$  be the child of  $v_j$  such that  $b \in \Lambda(T_3^{c_b})$ . Then  $\text{count}_{r,f,f}^{T_1}(a, b) = \sigma^{T_1, -T_2}(W_j, b) - \sigma^{T_1, -T_2}(\Lambda(T_3^{c_b}), b)$ .*

**Lemma 11.** *After  $O(n)$  time preprocessing, given any  $S \subseteq L$ ,  $\sigma^{T_1, -T_2}(S, b)$  for all  $b \in S$  can be computed in  $O(|S|)$  time.*

This suggests the following algorithm, which we call `Compute_count_rff_T1`, for computing  $\text{count}_{r,f,f}^{T_1}(a, b)$  for all  $b \in L \setminus \{a\}$  for any fixed  $a \in L$ . First, it builds the partition  $\{W_1, \dots, W_p\}$  of  $L \setminus \{a\}$ . This takes  $O(n)$  time. Then,  $T_1$  and  $T_2$  are preprocessed in  $O(n)$  time so Lemma 11 can be applied. For each  $j \in \{1, \dots, p\}$ , the algorithm then computes  $\sigma^{T_1, -T_2}(W_j, b)$  and  $\sigma^{T_1, -T_2}(\Lambda(T_3^{c_b}), b)$  for all  $b \in W_j$ . By Lemma 11, this step takes  $O(\sum_{j=1}^p |W_j|) = O(n)$  time. (For every  $b \in W_j$ , to identify the child  $c_b$  of  $v_j$  such that  $b \in \Lambda(T_3^{c_b})$  in  $O(1)$  time, one can store the depths of all nodes in  $T_3$  and use the level-ancestor data structure after  $O(n)$  time extra preprocessing [3].) Finally, Lemma 10 is used to obtain  $\text{count}_{r,f,f}^{T_1}(a, b)$  for every  $b \in W_j$  and  $j \in \{1, \dots, p\}$  in  $O(n)$  time. In total, the time complexity of `Compute_count_rff_T1` is  $O(n)$ . By running `Compute_count_rff_T1` once for each  $a \in L$ , we get  $\text{count}_{r,f,f}^{T_1}(a, b)$  for all  $a, b \in L$  in  $O(n^2)$  total time.

**Lemma 12.** *For each  $i \in \{1, 2, 3\}$ , the values of  $\text{count}_{r,f,f}^{T_i}(a, b)$  for all  $a, b \in L$  can be computed in  $O(n^2)$  total time.*

### 3.4 Determining if a Given Cluster Is a Strong Cluster When $k = 3$

Steps 3–5 of `R*.consensus_tree` in Section 1.3 need to determine which Apresjan clusters of  $s_{\mathcal{R}_{maj}}$  are strong clusters of  $\mathcal{R}_{maj}$ . This subsection presents a method for doing so efficiently. Let  $A \subseteq L$ . For any  $j \in \{1, 2, 3\}$ , a leaf  $x \in L \setminus A$  is called an *outsider* in  $T_j$  if  $x$  is not a descendant of  $u_A^j$  in  $T_j$ , where  $u_A^j = \text{lca}^{T_j}(A)$ . Define the following two disjoint subsets of  $L \setminus A$ : (i)  $P_A$  = the set of all  $x \in L \setminus A$  such that  $\text{lca}^{T_j}(a, x)$  is a proper descendant of  $u_A^j$  for some  $a \in A$  and some  $j \in \{1, 2, 3\}$ ; and (ii)  $Q_A$  = the set of all  $x \in L \setminus A$  such that  $\text{lca}^{T_j}(a, x) = u_A^j$  for all  $a \in A$  and all  $j \in \{1, 2, 3\}$ . (If  $|A| = 1$  then  $P_A = Q_A = \emptyset$ .) Also define an undirected graph  $G_A = (A, E_A)$ , whose edge set is  $E_A = \{\{a, a'\} : \text{lca}^{T_j}(a, a') \text{ is a proper descendant of } u_A^j \text{ for at least one } j \in \{1, 2, 3\}\}$ . Then we have:

**Lemma 13.** *For any  $A \subseteq L$ ,  $A$  is a strong cluster of  $\mathcal{R}_{maj}$  if and only if: (1) each  $x \in P_A$  is an outsider in exactly two trees from  $\{T_1, T_2, T_3\}$ ; and (2) if  $Q_A$  is nonempty, the graph  $G_A$  is a complete graph.*

```

Procedure Check_all_Apresjan_clusters
Input: A tree  $\mathcal{A}$  of all Apresjan clusters of  $s_{\mathcal{R}_{maj}}$ 
Output: A list of all the strong clusters of  $\mathcal{R}_{maj}$ 
1: for all nodes  $v$  in  $\mathcal{A}$  in bottom-up order do
2:   Let  $A$  be the Apresjan cluster of  $s_{\mathcal{R}_{maj}}$  corresponding to  $v$ ;
3:   if  $v$  is a leaf then
4:     /* Without loss of generality, assume  $A = \{a\}$  */
5:     Set  $u_A^1 = u_A^2 = u_A^3$  to be the leaf with label  $a$  and let  $G_A$  be a graph with
       a single vertex  $a$ . Let  $\mathcal{B}_A^1 = \mathcal{B}_A^2 = \mathcal{B}_A^3 = \{A\}$ ;
6:   else
7:     Let  $A_1, \dots, A_m$  be the Apresjan clusters corresponding to the children of  $v$ 
       and form  $G_A$  by merging  $G_{A_1}, \dots, G_{A_m}$ ;
8:     for  $j = 1, 2, 3$  do
9:       Update  $u_A^j = lca^{T_j}(u_{A_1}^j, \dots, u_{A_m}^j)$ . Partition  $A$  into a set of blocks  $\mathcal{B}_A^j$ 
       such that each block  $B \in \mathcal{B}_A^j$  contains all the elements of  $A$  that appear
       in the same subtree attached to  $u_A^j$ ;
10:      Compute  $Z_B = \bigcup_{i=1}^m (\mathcal{B}_{A_i}^j | B)$  for every block  $B \in \mathcal{B}_A^j$ ;
11:      for every block  $B \in \mathcal{B}_A^j$  do
12:        Insert all edges  $\{x, y\}$  into  $G_A$  where  $x \in X, y \in Y$  and where  $X$  and
         $Y$  are two different sets in  $Z_B$ ;
13:      end for
14:    end for
15:  end if
16:  If  $A$  satisfies the condition in Lemma 13 then output  $A$ ;
17: end for

```

**Fig. 3.** Procedure for finding all strong clusters of  $\mathcal{R}_{maj}$

Procedure `Check_all_Apresjan_clusters` in Fig. 3 applies the condition in Lemma 13 to find all strong clusters of  $\mathcal{R}_{maj}$ . To avoid building each  $G_A$ -graph from scratch, it assumes that the Apresjan clusters are specified in the form of a tree  $\mathcal{A}$ , so that the information in the  $G_A$ -graphs can be reused as it goes upwards in  $\mathcal{A}$ . (As mentioned in Section 1.3,  $\mathcal{A}$  can be obtained in  $O(n^2)$  time [5].) The procedure builds the  $G_A$ -graphs for all Apresjan clusters  $A$  bottom-up, according to the given tree  $\mathcal{A}$ . Each  $G_A$  is represented as a set of edges. To simplify the construction, for  $j = \{1, 2, 3\}$ , the procedure maintains  $u_A^j = lca^{T_j}(A)$ . It also maintains  $\mathcal{B}_A^j$ , which is the partition of  $A$  such that each block  $B \in \mathcal{B}_A^j$  contains all elements in  $A$  that appear in one subtree attached to the node  $u_A^j$ .

For any set  $\mathcal{X}$  of subsets of  $L$  and any  $L' \subseteq L$ , let  $\mathcal{X}|L' = \{X \in \mathcal{X} : X \subseteq L'\}$ .

**Lemma 14.** *Procedure `Check_all_Apresjan_clusters` outputs all strong clusters of  $\mathcal{R}_{maj}$  in  $O(n^2 \alpha(n))$  time, where  $\alpha(n)$  is the inverse Ackermann function.*

## 4 Computing the $R^*$ Consensus Tree for Unbounded $k$

Section 4.1 computes  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $a, b \in L$  in  $O(n^2 \log^k n)$  time. Section 4.2 checks which Apresjan clusters are strong clusters in  $O(n^2 \log^{k+2} n)$  time.

### 4.1 Computing $s_{\mathcal{R}_{maj}}$ for Unbounded $k$

Here, we give a procedure that, for any fixed  $a \in L$ , computes  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $b \in L \setminus \{a\}$  in  $O(n \log^k n)$  time.

Let  $occ(ab|w, T_{[i..j]})$  be the number of occurrences of  $ab|w$  in  $t(T_i), \dots, t(T_j)$ . Denote  $s_{T_{[1..i]}}^{W,x,y,z}(a, b) = |\{w \in W : occ(ab|w, T_{[1..i]}) + x > \max\{occ(aw|b, T_{[1..i]}) + y, occ(bw|a, T_{[1..i]}) + z\}|$ . For a fixed  $a \in L$ , our goal is to compute  $s_{\mathcal{R}_{maj}}(a, b) = s_{T_{[1..k]}}^{L,0,0,0}(a, b)$  for all  $b \in L \setminus \{a\}$ . Note that in the formula for  $s_{T_{[1..i]}}^{W,x,y,z}(a, b)$ ,  $W$  is not any arbitrary subset of  $L$ ; we require, for all  $w \in W$ , that  $x, y$  and  $z$  are the number of occurrences of  $ab|w, aw|b$  and  $bw|a$ , respectively, in  $T_{i+1}, \dots, T_k$ . These three integers will be used to pass information during recursive calls.

In each tree  $T_i \in \{T_1, \dots, T_k\}$ , any  $w \in L \setminus \{a\}$  is represented by a pair  $(d^{T_i}(w), \pi_i(w))$ , where  $d^{T_i}(w)$  is the distance in  $T_i$  from  $a$  to  $lca^{T_i}(a, w)$ , and  $\pi_i(w) = j$ , where  $w$  is a descendant of the  $j$ th child of  $lca^{T_i}(a, w)$ . The occurrence of a triplet in  $t(T_i)$  is then given by (cf. Theorem 1 in [12] and Lemma 7 in [10]):

**Lemma 15.** *Let  $b \in L \setminus \{a\}$ . For any  $w \in L \setminus \{a, b\}$  and  $i \in \{1, \dots, k\}$ :*

1.  $ab|w \in t(T_i)$  if and only if  $d^{T_i}(b) < d^{T_i}(w)$ ;
2.  $aw|b \in t(T_i)$  if and only if  $d^{T_i}(b) > d^{T_i}(w)$ ; and
3.  $bw|a \in t(T_i)$  if and only if  $d^{T_i}(b) = d^{T_i}(w)$  and  $\pi_i(b) = \pi_i(w)$ .

We build a data structure  $B_{W,k}$  in  $O(|W| \log^k |W|)$  time that yields the value of  $s_{T_{[1..k]}}^{W,x,y,z}(a, b)$  for any  $b \in W \setminus \{a\}$  and any  $x, y, z$  in  $O(\log^k |W|)$  time as follows.

For the base case  $k = 1$ , the data structure  $B_{W,1}$  consists of a balanced binary search tree  $BT(W, T_1)$  for all distinct  $d^{T_1}(w)$ -values, where  $w \in W$ . There may be multiple elements of  $W$  with the same  $d^{T_1}(w)$ -value. For each such node, we replace it by a balanced binary search tree for these multiple elements and index them using the keys  $\pi_1(w)$ . The additional nodes are called *yellow nodes*. The data structure  $B_{W,1}$  can be constructed in  $O(|W|)$  time.

Now we show how to compute  $s_{T_{[1..1]}}^{W,x,y,z}(a, b)$  from  $B_{W,1}$ . For any  $b \in W$ , let  $P$  be the path from the root of  $BT(W, T_1)$  to  $b$ . Since  $BT(W, T_1)$  is balanced,  $P$  is of length  $O(\log |W|)$ . We partition the subtrees attached to  $P$  into four sets:

- $W_{fan}$  is the set of subtrees attached to the yellow nodes of  $P$  where  $\pi_1(b) \neq \pi_1(w)$  for all leaves  $w$  in the subtrees of  $W_{fan}$ .
- $W_{mid}$  is the set of subtrees attached to the yellow nodes of  $P$  where  $\pi_1(b) = \pi_1(w)$  for all leaves  $w$  in the subtrees of  $W_{mid}$ .
- $W_{left}$  is the set of left subtrees attached to the non-yellow nodes of  $P$ .
- $W_{right}$  is the set of right subtrees attached to the non-yellow nodes of  $P$ .

Note that  $ab|w \in t(T_1)$  for all  $w \in A(S)$  and  $S \in W_{fan}$ . Similarly,  $bw|a \in t(T_1)$  for all  $w \in A(S)$  and  $S \in W_{mid}$ . Also,  $aw|b \in t(T_1)$  for all  $w \in A(S)$  and  $S \in W_{left}$ , and  $ab|w \in t(T_1)$  for all  $w \in A(S)$  and  $S \in W_{right}$ .

By the definitions and Lemma 15,  $s_{T_{[1..1]}}^{W,x,y,z}(a, b) = A + B + C + D$  where:

- $A = \sum_{S \in W_{fan}} |A(S)|$  if  $x > y, x > z$ ; and 0 otherwise.
- $B = \sum_{S \in W_{mid}} |A(S)|$  if  $x > y, x > 1 + z$ ; and 0 otherwise.
- $C = \sum_{S \in W_{left}} |A(S)|$  if  $x > 1 + y, x > z$ ; and 0 otherwise.
- $D = \sum_{S \in W_{right}} |A(S)|$  if  $x + 1 > y, x + 1 > z$ ; and 0 otherwise.

```

Procedure counting_query
Input: Integer  $i \in \{0, 1, \dots, k\}$ ,  $W \subseteq L$ , integers  $x, y, z$ , leaf  $b \in L \setminus \{a\}$ .
Output:  $s_{T_{[1..i]}}^{W,x,y,z}(a, b)$ 
1: if  $i = 0$  then
2:   if  $x > y$  and  $x > z$  then
3:     return  $|W|$ ;
4:   else
5:     return 0;
6:   end if
7: else
8:   Let  $P$  be the path from the root of  $BT(W, T_i)$  to  $b$ ;
9:   Compute the sets  $W_{fan}, W_{mid}, W_{right}, W_{left}$  of subtrees attached to  $P$ ;
10:   $A = \sum_{S \in W_{fan}} \text{counting\_query}(i - 1, \Lambda(S), x, y, z, b)$ ;
11:   $B = \sum_{S \in W_{mid}} \text{counting\_query}(i - 1, \Lambda(S), x, y, z + 1, b)$ ;
12:   $C = \sum_{S \in W_{left}} \text{counting\_query}(i - 1, \Lambda(S), x, y + 1, z, b)$ ;
13:   $D = \sum_{S \in W_{right}} \text{counting\_query}(i - 1, \Lambda(S), x + 1, y, z, b)$ ;
14:  return  $A + B + C + D$ ;
15: end if

```

**Fig. 4.** Procedure for computing  $s_{T_{[1..i]}}^{W,x,y,z}(a, b)$ , assuming  $B_{W,i}$  is available

There are  $O(\log |W|)$  subtrees, so we can find  $s_{T_{[1..i]}}^{W,x,y,z}(a, b)$  in  $O(\log |W|)$  time.

Next, assume we can create a data structure  $B_{W,k-1}$  from which  $s_{T_{[1..k-1]}}^{W,x,y,z}(a, b)$  can be computed in  $O(\log^{k-1} |W|)$  time. Then we build the data structure  $B_{W,k}$ , consisting of two parts, as follows. Firstly, similar to the case  $k = 1$ , we build a binary search tree  $BT(W, T_k)$ . Secondly, for every subtree  $S$  in  $BT(W, T_k)$ , we build the data structure  $B_{\Lambda(S),k-1}$ . The time required to build  $B_{W,k}$  depends on the time needed for the two parts. For the first part, as shown above,  $BT(W, T_k)$  can be constructed in  $O(|W| \log |W|)$  time. For the second part,  $\sum\{\Lambda(S) : S \text{ is a subtree of } BT(W, T_k)\} = O(|W| \log |W|)$ . Since  $B_{\Lambda(S),k-1}$  can be constructed in  $O(|\Lambda(S)| \log^{k-1} |\Lambda(S)|)$  time, the second part takes  $O(|W| \log^k |W|)$  time.

We now discuss how to use  $B_{W,k}$  to compute  $s_{T_{[1..k]}}^{W,x,y,z}(a, b)$ . For any  $b \in W$ , similar to the case  $k = 1$ , first find the path  $P$  from the root of  $BT(W, T_k)$  to  $b$ . There are  $O(\log |W|)$  subtrees attached to  $P$ . Partition them into the sets  $W_{fan}, W_{mid}, W_{left}$ , and  $W_{right}$  according to the same criteria as for  $k = 1$  above. Then:

**Lemma 16.** For any  $b \in W$ , it holds that  $s_{T_{[1..k]}}^{W,x,y,z}(a, b) = A + B + C + D$ , where  $A = \sum_{S \in W_{fan}} s_{T_{[1..k-1]}}^{\Lambda(S),x,y,z}(a, b)$ ,  $B = \sum_{S \in W_{mid}} s_{T_{[1..k-1]}}^{\Lambda(S),x,y,z+1}(a, b)$ ,  $C = \sum_{S \in W_{left}} s_{T_{[1..k-1]}}^{\Lambda(S),x,y+1,z}(a, b)$ , and  $D = \sum_{S \in W_{right}} s_{T_{[1..k-1]}}^{\Lambda(S),x+1,y,z}(a, b)$ .

Fig. 4 lists the pseudocode of the procedure `counting_query` for computing  $s_{T_{[1..k]}}^{W,x,y,z}(a, b)$ , given  $B_{W,k}$ . The next lemma bounds its running time.

**Lemma 17.** Given the data structure  $B_{W,k}$  for a fixed  $a \in L$ , for any  $b \in L \setminus \{a\}$ , `counting_query`( $k, W, x, y, z, b$ ) computes  $s_{T_{[1..k]}}^{W,x,y,z}(a, b)$  in  $O(\log^k n)$  time.

### 4.2 Determining if a Given Cluster Is a Strong Cluster for Unbounded $k$

Let  $\mathcal{A}$  be the tree of all Apresjan clusters. For any  $A \subseteq L$  and  $a, b \in A$  with  $a \neq b$ , define  $s_{\mathcal{R}_{maj}}^A(a, b) = |\{w \in A : ab|w \in \mathcal{R}_{maj}\}|$ . The following lemma allows us to verify if  $A$  is a strong cluster.

**Lemma 18.** *Let  $A \subseteq L$ .  $A$  is a strong cluster of  $\mathcal{R}_{maj}$  if and only if  $s_{\mathcal{R}_{maj}}(a, b) = |L \setminus A| + s_{\mathcal{R}_{maj}}^A(a, b)$  for all  $a, b \in A$  with  $a \neq b$ .*

Observe that  $s_{\mathcal{R}_{maj}}^A(a, b) = s_{T_{[1..k]}}^{A,0,0,0}(a, b)$ , using the notation from Section 4.1. For any fixed  $a \in L$ , the next lemma gives a data structure for computing  $s_{\mathcal{R}_{maj}}^A(a, b)$  in  $O(\log^{k+1} n)$  time for any cluster  $A \in \mathcal{A}$  and  $b \in A \setminus \{a\}$ .

**Lemma 19.** *For any  $a \in L$ , we can construct a data structure in  $O(n \log^{k+1} n)$  time which enables us to compute  $s_{\mathcal{R}_{maj}}^A(a, b) = s_{T_{[1..k]}}^{A,0,0,0}(a, b)$  in  $O(\log^{k+1} n)$  time for any cluster  $A \in \mathcal{A}$  that contains the element  $a$  and any  $b \in A \setminus \{a\}$ .*

**Lemma 20.** *If a node  $u$  in  $\mathcal{A}$  satisfies  $s_{\mathcal{R}_{maj}}(a, b) = |L \setminus \Lambda(\mathcal{A}^u)| + s_{\mathcal{R}_{maj}}^{A(\mathcal{A}^u)}(a, b)$ , then, for every ancestor  $u'$  of  $u$ ,  $s_{\mathcal{R}_{maj}}(a, b) = |L \setminus \Lambda(\mathcal{A}^{u'})| + s_{\mathcal{R}_{maj}}^{A(\mathcal{A}^{u'})}(a, b)$  holds.*

Thus,  $\mathcal{A}$  contains a node  $u_{min}^{a,b}$  such that  $s_{\mathcal{R}_{maj}}(a, b) = |L \setminus \Lambda(\mathcal{A}^u)| + s_{\mathcal{R}_{maj}}^{A(\mathcal{A}^u)}(a, b)$  for any ancestor  $u$  of  $u_{min}^{a,b}$ . In fact,  $u_{min}^{a,b}$  can be found in  $O(\log^{k+2} n)$  time:

**Lemma 21.** *Given the data structure in Lemma 19,  $u_{min}^{a,b}$  for any  $b \in L$  can be found in  $O(\log^{k+2} n)$  time.*

Finally, we describe the procedure `Verify_strong_clusters` for checking which clusters in  $\mathcal{A}$  are strong clusters. See Fig. 5 for the pseudocode. First, initialize  $count(u) = 0$  for every node  $u$  in  $\mathcal{A}$ . Then, compute  $u_{min}^{a,b}$  for all  $a, b \in L$  using Lemma 21, and increase

```

Procedure Verify_strong_clusters
Input: A tree  $\mathcal{A}$  of all Apresjan clusters of  $\mathcal{R}_{maj}$ 
Output: A tree including all strong clusters of  $\mathcal{R}_{maj}$ 
1: Set  $count(u) = 0$  for all nodes  $u$  in  $\mathcal{A}$ ;
2: for  $a, b \in L$  do
3:   Find  $u_{min}^{a,b}$  by Lemma 21 and set  $count(u_{min}^{a,b}) = count(u_{min}^{a,b}) + 1$ ;
4: end for
5: Set  $sum(u) = 0$  for all leaves  $u$  in  $\mathcal{A}$ ;
6: for every internal node  $u \in \mathcal{A}$  in bottom-up order do
7:   Set  $sum(u) = count(u) + \sum\{sum(c) : c \text{ is a child of } u \text{ in } \mathcal{A}\}$ ;
8:   if  $sum(u) < \binom{|\Lambda(\mathcal{A}^u)|}{2}$  then
9:     Contract node  $u$ ; /*  $\Lambda(\mathcal{A}^u)$  is not a strong cluster */
10:  end if
11: end for
12: return  $\mathcal{A}$ ;
    
```

**Fig. 5.** Procedure for checking which Apresjan clusters are strong clusters

each  $count(u_{min}^{a,b})$  by 1. Next, set  $sum(u)$  to be the total sum of  $count(v)$  for all descendants  $v$  of  $u$  in  $\mathcal{A}$ . By Lemma 22 below, if  $sum(u) = \binom{|\Lambda(\mathcal{A}^u)|}{2}$  then  $\Lambda(\mathcal{A}^u)$  is a strong cluster; otherwise, it is not. In case  $\Lambda(\mathcal{A}^u)$  is not a strong cluster, contract  $u$  in  $\mathcal{A}$  (that is, attach all children of  $u$  to the parent of  $u$  in  $\mathcal{A}$  and remove the node  $u$ ). By Lemmas 19 and 21, the running time of `Verify_strong_clusters` is  $O(n^2 \log^{k+2} n)$ .

**Lemma 22.** *For any node  $u$  in  $\mathcal{A}$ ,  $\Lambda(\mathcal{A}^u)$  is a strong cluster if and only if  $sum(u) = \binom{|\Lambda(\mathcal{A}^u)|}{2}$ .*

## References

1. Bansal, M.S., Dong, J., Fernández-Baca, D.: Comparing and aggregating partially resolved trees. *Theoretical Computer Science* **412**(48), 6634–6652 (2011)
2. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) *LATIN 2000*. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
3. Bender, M.A., Farach-Colton, M.: The Level Ancestor Problem simplified. *Theoretical Computer Science* **321**(1), 5–12 (2004)
4. Bryant, D.: A classification of consensus methods for phylogenetics. In: Janowitz, M.F., Lapointe, F.-J., McMorris, F.R., Mirkin, B., Roberts, F.S. (eds.) *Bioconsensus*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 61, pp. 163–184. American Mathematical Society (2003)
5. Bryant, D., Berry, V.: A structured family of clustering and tree construction methods. *Advances in Applied Mathematics* **27**(4), 705–732 (2001)
6. Chan, T.M., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pp. 161–173. SIAM (2010)
7. Degnan, J.H., DeGiorgio, M., Bryant, D., Rosenberg, N.A.: Properties of consensus methods for inferring species trees from gene trees. *Systematic Biology* **58**(1), 35–54 (2009)
8. Felsenstein, J.: *Inferring Phylogenies*. Sinauer Associates Inc., Sunderland (2004)
9. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing* **13**(2), 338–355 (1984)
10. Jansson, J., Sung, W.-K.: Constructing the R\* consensus tree of two trees in sub-cubic time. *Algorithmica* **66**(2), 329–345 (2013)
11. Kannan, S., Warnow, T., Yooshef, S.: Computing the local consensus of trees. *SIAM Journal on Computing* **27**(6), 1695–1724 (1998)
12. Lee, C.-M., Hung, L.-J., Chang, M.-S., Shen, C.-B., Tang, C.-Y.: An improved algorithm for the maximum agreement subtree problem. *Information Processing Letters* **94**(5), 211–216 (2005)
13. Sung, W.-K.: *Algorithms in Bioinformatics: A Practical Introduction*. Chapman & Hall/CRC (2010)

# **Fixed-Parameter Tractable Algorithms II**



# Complexity and Kernels for Bipartition into Degree-bounded Induced Graphs

Mingyu Xiao<sup>1</sup>(✉) and Hiroshi Nagamochi<sup>2</sup>

<sup>1</sup> School of Computer Science and Engineering,  
University of Electronic Science and Technology of China, Sichuan, China  
myxiao@gmail.com

<sup>2</sup> Department of Applied Mathematics and Physics, Graduate School of Informatics,  
Kyoto University, Kyoto, Japan  
nag@amp.i.kyoto-u.ac.jp

**Abstract.** In this paper, we study the parameterized complexity of the problems of partitioning the vertex set of a graph into two parts  $V_A$  and  $V_B$  such that  $V_A$  induces a graph with degree at most  $a$  (resp., an  $a$ -regular graph) and  $V_B$  induces a graph with degree at most  $b$  (resp., a  $b$ -regular graph). These two problems are called UPPER-DEGREE-BOUNDED BIPARTITION and REGULAR BIPARTITION respectively. First, we prove that the two problems are NP-complete with any nonnegative integers  $a$  and  $b$  except  $a = b = 0$ . Second, we show that the two problems with parameter  $k$  being the size of  $V_A$  of a bipartition  $(V_A, V_B)$  are fixed-parameter tractable for fixed integer  $a$  or  $b$  by deriving some problem kernels for them.

## 1 Introduction

In graph algorithms and graph theory, there is a series of important problems that require us to partition the vertex set of a graph into several parts such that each part induces a subgraph satisfying some degree constraints. For example, the  $k$ -coloring problem is to partition the graph into  $k$  parts each of which induces an independent set (a 0-regular graph). Most of these kinds of problems are NP-hard, even if the problems are to partition a given graph into only two parts, which is called a *bipartition*.

For bipartition with a degree constraint on each part, we can find many references related to this topic. Here is a definition of the problem:

---

DEGREE-CONSTRAINED BIPARTITION

**Instance:** A graph  $G = (V, E)$  and four integers  $a, a', b$  and  $b'$ .

**Question:** Is there a partition  $(V_A, V_B)$  of  $V$  such that

$$a' \leq \deg_{V_A}(v) \leq a \quad \forall v \in V_A \quad \text{and} \quad b' \leq \deg_{V_B}(v) \leq b \quad \forall v \in V_B,$$

where  $\deg_X(v)$  denotes the degree of a vertex  $v$  in the induced subgraph  $G[X]$ ?

There are three special cases of DEGREE-CONSTRAINED BIPARTITION. If there are no constraints on the upper bounds (resp., lower bounds) of the degree in DEGREE-CONSTRAINED BIPARTITION, i.e.,  $a = b = \infty$  (resp.,  $a' = b' = 0$ ), we call the problem LOWER-DEGREE-BOUNDED BIPARTITION (resp., UPPER-DEGREE-BOUNDED BIPARTITION). We call DEGREE-CONSTRAINED BIPARTITION with a special case of  $a = a'$  and  $b = b'$  REGULAR BIPARTITION.

LOWER-DEGREE-BOUNDED BIPARTITION has been extensively studied in the literature. The problem with 4-regular graphs is NP-complete for  $a' = b' = 3$  [7] and linear-time solvable for  $a' = b' = 2$  [4]. More polynomial-time solvable cases with restrictions on the structure of given graphs and constraints on  $a'$  and  $b'$  can be found in [2, 3, 7, 12, 16].

For REGULAR BIPARTITION, when  $a = b = 0$ , the problem becomes a polynomial-solvable problem of checking whether a given graph is bipartite or not; when  $a = 0$  and  $b = 1$ , the problem becomes DOMINATING INDUCED MATCHING, a well studied NP-hard problem also known as EFFICIENT EDGE DOMINATION [11, 14]. However, not many results are known about UPPER-DEGREE-BOUNDED BIPARTITION and REGULAR BIPARTITION with other values of  $a$  and  $b$ .

In this paper, we first show that UPPER-DEGREE-BOUNDED BIPARTITION and REGULAR BIPARTITION are NP-complete with any nonnegative integers  $a$  and  $b$  except  $a = b = 0$ . The major contributions of this paper are vertex kernels for these two problems, which also implies that for constants  $a$  and  $b$  they are fixed-parameter tractable (FPT) with parameter  $k = |V_A|$ . We also discuss the fixed-parameter intractability of our problems with parameter only  $k = |V_A|$  where  $b$  is not fixed.

We also note some related problems, in which the degree constraint on one part of the bipartition changes to a constraint on the size of the part. BOUNDED-DEGREE DELETION asks us to delete at most  $k$  vertices from a graph to make the remaining graph having maximum vertex degree at most  $a$ . MAXIMUM REGULAR INDUCED SUBGRAPH asks us to delete at most  $k$  vertices from a graph to make the remaining graph an  $a$ -regular graph. These two problems can be regarded as such a kind of bipartition problems and have been well studied in parameterized complexity. They are FPT with parameters  $k$  and  $a$  and W[1]-hard with only parameter  $k$  [10, 15, 16]. Let  $tw$  denote the treewidth of an input graph. Betzler et. al. also proved that BOUNDED-DEGREE DELETION is FPT with parameters  $k$  and  $tw$  and W[2]-hard with only parameter  $tw$  [6]. The parameterized complexity of some other related problems, such as MINIMUM REGULAR INDUCED SUBGRAPH are studied in [1].

The remaining parts of the paper are organized as follows: Section 2 introduces a notation system. Section 3 proves the NP-hardness of our problems. Section 4 gives the problem kernels, and Section 5 shows the fixed-parameter intractability. Finally, some concluding remarks are given in the last section. Proofs of some lemmas are omitted due to space limitation.

## 2 Preliminaries

In this paper, a graph stands for a simple undirected graph. We may simply use  $v$  to denote the set  $\{v\}$  of a single vertex  $v$ . Let  $G = (V, E)$  be a graph, and  $X \subseteq V$  be a subset of vertices. The subgraph induced by  $X$  is denoted by  $G[X]$ , and  $G[V \setminus X]$  is also written as  $G \setminus X$ . Let  $E(X)$  denote the set of edges between  $X$  and  $V \setminus X$ . Let  $N(X)$  denote the *neighbors* of  $X$ , i.e., the vertices  $y \in V \setminus X$  adjacent to a vertex  $x \in X$ , and denote  $N(X) \cup X$  by  $N[X]$ . The *degree*  $\deg(v)$  of a vertex  $v$  is defined to be  $|N(v)|$ . A vertex in  $X$  is called an  $X$ -*vertex*, and a neighbor  $u \in X$  of a vertex  $v$  is called an  $X$ -*neighbor* of  $v$ . The number of  $X$ -neighbors of  $v$  is denoted by  $\deg_X(v)$ ; i.e.,  $\deg_X(v) = |N(v) \cap X|$ . The vertex set and edge set of a graph  $H$  are denoted by  $V(H)$  and  $E(H)$ , respectively. When  $X$  is equal to  $V(H)$  of some subgraph  $H$  of  $G$ , we may denote  $V(H)$ -vertices by  $H$ -vertices,  $V(H)$ -neighbors by  $H$ -neighbors, and  $\deg_{V(H)}(v)$  by  $\deg_H(v)$  for simplicity. For a subset  $E' \subseteq E$ , let  $G - E'$  denote the subgraph obtained from  $G$  by deleting edges in  $E'$ . For an integer  $p \geq 1$ , a star with  $p + 1$  vertices is called a  $p$ -*star*. The unique vertex of degree  $> 1$  in a  $p$ -star with  $p > 1$  is called the *center* of the star, and any vertex in a 1-star is a *center* of the star.

For a graph  $G$  and two nonnegative integers  $a$  and  $b$ , a partition of  $V(G)$  into  $V_A$  and  $V_B$  is called  $(a, b)$ -*bounded* if  $\deg_{V_A}(v) \leq a$  for all vertices in  $v \in V_A$  and  $\deg_{V_B}(v) \leq b$  for all vertices in  $v \in V_B$ . An  $(a, b)$ -bounded partition  $(V_A, V_B)$  is called  $(a, b)$ -*regular* if  $\deg_{V_A}(v) = a$  for all vertices in  $v \in V_A$  and  $\deg_{V_B}(v) = b$  for all vertices in  $v \in V_B$ . An instance  $I = (G, a, b)$  of UPPER-DEGREE-BOUNDED BIPARTITION (resp., REGULAR BIPARTITION) consists of a graph  $G$  and two nonnegative integers  $a$  and  $b$ , and asks us to test whether an instance  $(G, a, b)$  admits an  $(a, b)$ -bounded partition (resp.,  $(a, b)$ -regular partition) or not.

## 3 NP-Hardness

**Theorem 1.** UPPER-DEGREE-BOUNDED BIPARTITION is NP-complete for any nonnegative integers  $a$  and  $b$  except  $a = b = 0$ .

Before proving Theorem 1, we first provide some properties on complete graphs in UPPER-DEGREE-BOUNDED BIPARTITION. Without loss of generality we assume that  $a \leq b$  and  $b \geq 1$  in this section.

An  $(a + 1, b + 1, a + 1)$ -*complete graph*  $W$  is defined to be the graph consisting of two complete graphs of size  $a + b + 2$  that share exactly  $b + 1$  vertices, where  $|V(W)| = 2(a + b + 2) - (b + 1) = 2a + b + 3$  holds and the set of  $b + 1$  vertices shared by the two complete graphs is denoted by  $S(W)$ .

**Lemma 1.** Let  $(G, a, b)$  admit an  $(a, b)$ -bounded partition  $(V_A, V_B)$ .

- (i) If  $G$  contains a clique  $K$  of size  $a + b + 2$ , then  $|V(K) \cap V_A| = a + 1$  and  $|V(K) \cap V_B| = b + 1$ ; and
- (ii) Assume that  $G$  contains an  $(a + 1, b + 1, a + 1)$ -complete graph  $W$ . Then  $\{V(W) \cap V_A, V(W) \cap V_B\} = \{S(W), V(W) \setminus S(W)\}$  (or  $V(W) \cap V_A = V(W) \setminus S(W)$  and  $V(W) \cap V_B = S(W)$  when  $a \neq b$ ),  $N(V_A \cap V(W)) \setminus V(W) \subseteq V_B$  and  $N(V_B \cap V(W)) \setminus V(W) \subseteq V_B$ .

We here construct a special graph that consists of an  $(a + 1, b + 1, a + 1)$ -complete graph, several complete graphs with size  $a + b + 2$  and some edges joining them. Given two positive integers  $n$  and  $m$ , we first construct an  $(a + 1, b + 1, a + 1)$ -complete graph  $W$  and  $(n + m)$  complete graphs  $X_1, X_2, \dots, X_n$  and  $C_1, C_2, \dots, C_m$  with size  $a + b + 2$ . Next we choose a vertex  $v_A \in V(W) \setminus S(W)$  and a vertex  $v_B \in S(W)$  arbitrarily, and add edges between  $\{v_A, v_B\}$  and  $\{X_1, \dots, X_i, \dots, X_n\} \cup \{C_1, \dots, C_j, \dots, C_m\}$  as follows:

1. For each  $X_i$ , join  $v_B$  to arbitrary  $a$  vertices  $u_1, \dots, u_a \in V(X_i)$  via new edges, and join  $v_A$  to arbitrary  $b$  vertices  $u'_1, \dots, u'_b \in V(X_i) \setminus \{u_1, \dots, u_a\}$  via new edges;
2. For each  $C_j$ , join  $v_B$  to arbitrary  $a$  vertices  $u_1, \dots, u_a \in V(C_j)$  via new edges, and join  $v_A$  to arbitrary  $(b - 1)$  vertices  $u'_1, \dots, u'_{b-1} \in V(C_j) \setminus \{u_1, \dots, u_a\}$  via new edges, where  $b - 1 \geq 0$  since  $b \geq 1$  is assumed; and
3. Let  $G_{n,m}$  denote the resulting graph.

Vertices in  $X_i$  ( $i = 1, 2, \dots, n$ ) or  $C_j$  ( $j = 1, 2, \dots, m$ ) not adjacent to  $v_A$  or  $v_B$  are called *free*. Each  $X_i$  contains exactly two free vertices, denoted by  $v_i$  and  $v'_i$ , and each  $C_j$  contains exactly three free vertices, denoted by  $v_j^1, v_j^2$  and  $v_j^3$ .

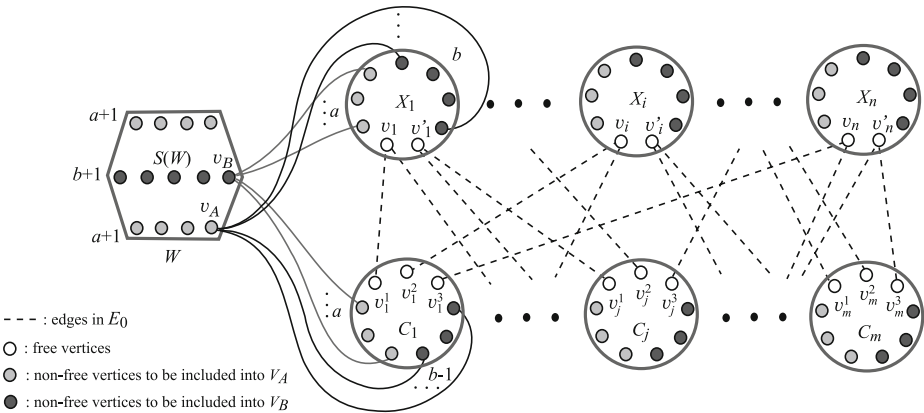


Fig. 1. Constructing graph  $G_{n,m} + E_0$

Let  $E_0$  be an arbitrary set of new edges between free vertices in  $\cup_{1 \leq i \leq n} X_i$  and free vertices in  $\cup_{1 \leq j \leq m} C_j$  in  $G_{n,m}$ . Let  $G_{n,m} + E_0$  be the graph obtained from  $G_{n,m}$  by adding the edges in  $E_0$ . See Figure 1. We have

**Lemma 2.** *Let  $(V_A, V_B)$  be a partition of  $V(G_{n,m} + E_0)$ , where if  $a = b$  then we assume without loss of generality that  $v_A \in V_A$ . Then  $(V_A, V_B)$  is an  $(a, b)$ -bounded partition of  $G_{n,m} + E_0$  if and only if the following hold:*

- (i) Every subgraph  $H \in \{W, X_1, \dots, X_n, C_1, \dots, C_m\}$  satisfies that  $\deg_{V(H) \cap V_A}(v) = a$  for all vertices  $v \in V(H) \cap V_A$  and  $\deg_{V(H) \cap V_B}(v) = b$  for all vertices  $v \in V(H) \cap V_B$ ;
- (ii)  $S(W) \subseteq V_B$ ,  $V(W) \setminus S(W) \subseteq V_A$ ,  $N(v_B) \setminus V(W) \subseteq V_A$ , and  $N(v_A) \setminus V(W) \subseteq V_B$ ;
- (iii) For each  $X_i$ , exactly one of the two free vertices in  $X_i$  is contained in  $V_A$  and the other is in  $V_B$ ; and
- (iv) For each  $C_j$ , exactly one of the three free vertices in  $C_j$  is contained in  $V_A$  and the other two are in  $V_B$ ; and (v) For each  $uv \in E_0$ ,  $|\{u, v\} \cap V_A| = |\{u, v\} \cap V_B| = 1$ .

Now we are ready to prove Theorem 1. Clearly UPPER-DEGREE-BOUNDED BIPARTITION is in NP. In what follows, we construct a polynomial reduction from the NP-complete problem ONE-IN-THREE 3SAT [12].

ONE-IN-THREE 3SAT

**Instance:** A set  $\mathcal{C}$  of  $m$  clauses  $c_1, c_2, \dots, c_m$  on a set  $\mathcal{X}$  of  $n$  variables  $x_1, x_2, \dots, x_n$  such that each clause  $c_j$  consists of exactly three literals  $\ell_j^1, \ell_j^2$  and  $\ell_j^3$ .

**Question:** Is there a truth assignment  $\mathcal{X} \rightarrow \{\text{true}, \text{false}\}^n$  such that each clause  $c_j$  has exactly one true literal?

Given an instance  $F = (\mathcal{C}, \mathcal{X})$  of ONE-IN-THREE 3SAT and nonnegative integers  $a \leq b$  ( $\geq 1$ ), we will construct an instance  $I_F = (G_F, a, b)$  of UPPER-DEGREE-BOUNDED BIPARTITION such that  $I_F$  has an  $(a, b)$ -bounded partition if and only if  $F$  is feasible. Such an instance  $I_F$  is constructed on the graph  $G_{n,m}$  by setting  $G_F = G_{n,m} + E_0$ , where a set  $E_0$  of edges between  $\{X_1, \dots, X_i, \dots, X_n\}$  and  $\{C_1, \dots, C_j, \dots, C_m\}$  according to the relationship between  $\mathcal{X}$  and  $\mathcal{C}$  in  $F$  as follows:

For each clause  $c_j = (\ell_j^1, \ell_j^2, \ell_j^3) \in \mathcal{C}$  and the  $k$ -th literal  $\ell_j^k$ ,  $k = 1, 2, 3$ , if  $\ell_j^k$  is a positive (resp., negative) literal of a variable  $x_i$ , then join free vertex  $v_j^k \in V(C_j)$  to free vertex  $v_i \in V(X_i)$  (resp.,  $v'_i \in V(X_i)$ ) via a new edge.

Let  $G_F = G_{n,m} + E_0$  be the resulting graph. We remark that  $X_i$  serves as a gadget for variable  $x_i \in \mathcal{X}$  and  $C_j$  serves as a gadget for clause  $c_j \in \mathcal{C}$ .

This completes the construction of instance  $I_F = (G_F, a, b)$ . We interpret conditions (iii) and (iv) on free vertices in Lemma 2 as follows:

$$v_i \in V_B \text{ (resp., } v_i \in V_A) \Leftrightarrow \text{true (resp., false) is assigned to } x_i, \text{ and}$$

$$v_j^k \in V_A \text{ (resp., } v_j^k \in V_B) \Leftrightarrow \ell_j^k = \text{true (resp., } \ell_j^k = \text{false).$$

Hence we see by Lemma 2 that  $I_F = (G_F = G_{n,m} + E_0, a, b)$  admits an  $(a, b)$ -bounded partition if and only if  $F$  is feasible. This completes a proof of Theorem 1.

By Lemma 2,  $F$  is feasible if and only if  $I_F = (G_F = G_{n,m} + E_0, a, b)$  admits an  $(a, b)$ -regular partition. Hence the problem of testing whether an instance  $(G, a, b)$  admits an  $(a, b)$ -regular partition is also NP-complete for any nonnegative integers  $a$  and  $b$  except  $a = b = 0$ .

**Corollary 1.** REGULAR BIPARTITION is NP-complete for any nonnegative integers  $a$  and  $b$  except  $a = b = 0$ .

## 4 Kernelization

This section studies the parameterized complexity and kernels of our problems. For this, we introduce the following constrained versions of the problems.

### CONSTRAINED UPPER-DEGREE-BOUNDED BIPARTITION

**Instance:** A graph  $G$ , two subsets  $A, B \subseteq V(G)$ , and nonnegative integers  $a, b$  and  $k$ .

**Question:** Is there an  $(a, b)$ -bounded partition  $(V_A, V_B)$  of  $V(G)$  such that  $A \subseteq V_A, B \subseteq V_B$ , and  $|V_A| \leq k$ ?

In the same way, we can define CONSTRAINED REGULAR BIPARTITION by replacing “ $(a, b)$ -bounded partition” with “ $(a, b)$ -regular partition” in the above definition. Note that we do not assume  $a \leq b$  in this section. We call a partition  $(V_A, V_B)$  satisfying the condition in the definitions of CONSTRAINED UPPER-DEGREE-BOUNDED BIPARTITION and CONSTRAINED REGULAR BIPARTITION a *solution* to the problem instance. An instance  $(G, A, B, a, b, k)$  is called *feasible* if it admits a solution. A vertex in  $V(G) \setminus (A \cup B)$  is called *undecided*, and we always denote  $V(G) \setminus (A \cup B)$  by  $U$ . Clearly each of the two problems can be solved in  $2^{|U|}|V|^{O(1)}$  time. We say that an instance  $(G, A, B, a, b, k)$  is *reduced* to an instance  $(G, A', B', a, b, k)$  such that  $(G, A, B, a, b, k)$  is feasible if and only if so is  $(G, A', B', a, b, k)$ . Note that when it turns out that  $(G, A, B, a, b, k)$  is infeasible we can say that it is reduced to an infeasible instance  $(G, A', B', a, b, k)$  such as one with  $A' \cap B' \neq \emptyset$ .

In this paper, we say that a problem admits a kernel of size  $O(f(k))$  if any instance of the problem can be reduced in polynomial time in  $n$  into an instance  $(G, A, B, a, b, k)$  with  $|V(G)| = O(f(k))$  for a function  $f(k)$  of  $k$ . The main results in this section are the following.

**Theorem 2.** CONSTRAINED UPPER-DEGREE-BOUNDED BIPARTITION admits a kernel of size  $O((b + 1)^2(b + k)k)$ , and is fixed-parameter tractable with parameter  $k$  for a constant  $b$ .

**Theorem 3.** CONSTRAINED REGULAR BIPARTITION admits a kernel of size  $O((b + 1)(b + k)k^2)$  for  $a \leq b$  or of size  $O((b + 1)(b + k)k^2 + (ak)^{(a-b+1)k})$  for  $a > b$ , and is fixed-parameter tractable with parameter  $k$  for constants  $a$  and  $b$ .

### 4.1 Kernels for Constrained Upper-Degree-Bounded Bipartition

In this subsection, an instance always means the one of CONSTRAINED UPPER-DEGREE-BOUNDED BIPARTITION. We have only five simple reduction rules to get a kernel to this problem.

**Rule 1.** Conclude that an instance is infeasible if one of the following holds:  $A \cap B \neq \emptyset$ ;  $|A| > k$ ;  $\deg_A(v) > a$  for some vertex  $v \in A$ ; and  $\deg_B(u) > b$  for some vertex  $u \in B$ .

**Rule 2.** Move to  $B$  any  $U$ -vertex  $v$  with  $\deg_A(v) > a$ , and move to  $A$  any  $U$ -vertex  $u$  with  $\deg_B(u) > b$ .

If we include to  $B$  a  $U$ -vertex  $v$  with  $\deg(v) > b+k$ , then the instance cannot have a solution, because at least  $k+1$  neighbors of  $v$  need to be included to  $A$ , implying that  $|V_A|$  cannot be bounded by  $k$ .

**Rule 3.** Move to  $A$  any  $U$ -vertex  $v$  with  $\deg(v) > b+k$ .

**Lemma 3.** Let  $v$  be a  $U \cup B$ -vertex in an instance  $I = (G, A, B, a, b, k)$  such that  $\deg(u) \leq b$  for all vertices  $u \in N[v]$ . Let  $I' = (G - \{v\}, A, B', a, b, k)$  be the instance obtained from  $I$  by deleting the vertex  $v$ , where  $B' = B$  if  $v \in U$  and  $B' = B - \{v\}$  if  $v \in B$ . The instance  $I$  is feasible if and only if so is  $I'$ .

**Proof.** It is clear that if  $I$  has a solution then  $I'$  also has a solution, because deleting a vertex never increases the degree of any of the remaining vertices. Assume that  $I'$  admits a solution  $(V_A, V_B)$ . We show that  $(V_A, V_B \cup \{v\})$  is a solution to  $I$ . Note that adding  $v$  to  $V_B$  may increase the degree of a vertex only in  $N[v]$ . However, by the choice of the vertex  $v$ , for any vertex  $u \in N[v]$  it holds  $b \geq \deg(u) \geq \deg_{V_B \cup \{v\}}(u)$ . Hence  $(V_A, V_B \cup \{v\})$  is a solution to  $I$ .  $\square$

**Rule 4.** Remove from the graph of an instance any  $U \cup B$ -vertex  $v$  such that  $\deg(u) \leq b$  for all vertices  $u \in N[v]$ .

**Lemma 4.** An instance  $I = (G, A, B, a, b, k)$  is infeasible if  $G$  contains more than  $k$  vertex-disjoint  $(b+1)$ -stars.

**Proof.** For a solution  $(V_A, V_B)$  to  $I$ , if there is a  $(b+1)$ -star disjoint with  $V_A$ , then a center  $v$  of the star would satisfy  $\deg_{V_B}(v) \geq b+1$ . Hence  $V_A$  must contain at least one from each of more than  $k$  vertex-disjoint  $(b+1)$ -stars. This, however, contradicts  $|V_A| \leq k$ .  $\square$

**Rule 5.** Compute a maximal set  $\mathcal{S}$  of vertex-disjoint  $(b+1)$ -stars in  $G$  of an instance  $I = (G, A, B, a, b, k)$  (not only in  $G[U]$ ). Conclude that the instance is infeasible if  $|\mathcal{S}| > k$ .

Now we analyze the size  $|V(G)|$  of an instance  $I = (G, A, B, a, b, k)$  where none of the above five rules can be applied anymore. Assume that when Rule 4 is applied to a maximal set of vertex-disjoint  $(b+1)$ -stars  $\mathcal{S}$  in  $G$ , it holds  $|\mathcal{S}| \leq k$  now. Let  $S_0$  be the set of all vertices in  $\mathcal{S}$ ,  $S_1 = N(S_0)$  and  $S_2 = N(S_1 \cup S_0) = N(S_1) \setminus S_0$ . We first show that  $V(G) = A \cup S_0 \cup S_1 \cup S_2$ . By the maximality of  $\mathcal{S}$ , we know that there is no vertex of degree  $\geq b+1$  in the graph after deleting  $S_0$ . Then all vertices  $u$  with  $\deg(u) \geq b+1$  are in  $S_0 \cup S_1$ , and  $|S_2| \leq b|S_1|$  holds. Since Rule 4 is no longer applicable, each  $U \cup B$ -vertex  $v$  with  $\deg(v) \leq b$  is adjacent to a vertex  $u$  with  $\deg(u) \geq b+1$  that is in

$S_0 \cup S_1$ . Then all  $U \cup B$ -vertices  $u$  with  $\deg(u) \leq b$  are in  $S_1 \cup S_2$ . Hence  $V(G) = A \cup S_0 \cup S_1 \cup S_2$ . We have that  $|A| \leq k$ ,  $|S_0| \leq (b + 2)|S| \leq (b + 2)k$ ,  $|S_1| \leq (b + k)|S_0| \leq (b + k)(b + 2)k$  by Rule 3 and  $|S_2| \leq b|S_1| \leq b(b + k)(b + 2)k$ . Therefore  $|V(G)| \leq |A| + |S_0| + |S_1| + |S_2| = O((b + 1)^2(b + k)k)$ . This proves Theorem 2.

### 4.2 Kernels for Constrained Regular Bipartition

In this subsection, an instance always stands for the one in CONSTRAINED REGULAR BIPARTITION. When we introduce a reduction rule, we assume that all previous reduction rules cannot be applied anymore.

We see that an instance  $I = (G, A, B, a, b)$  is infeasible if one of the following conditions holds:

- (i)  $A \cap B \neq \emptyset$  or  $|A| > k$ ;
- (ii) There is a vertex  $v \in V(G)$  with  $\deg(v) < \min\{a, b\}$ ;
- (iii) There is a vertex  $v \in A$  with  $\deg_{V(G) \setminus B}(v) < a$  or  $\deg_A(v) > a$ ; and
- (iv) There is a vertex  $v \in B$  with  $\deg_{V(G) \setminus A}(v) < b$  or  $\deg_B(v) > b$ .

**Rule 6.** Conclude that an instance is infeasible if one of the above four conditions holds.

**Rule 7.** Move to  $B$  any  $U$ -vertex  $v$  with  $\deg_{V(G) \setminus B}(v) < a$  or  $\deg_A(v) > a$  or adjacent to a  $B$ -vertex  $u$  with  $\deg_B(u) + \deg_U(u) = b$ . Move to  $A$  any  $U$ -vertex  $v$  with  $\deg_{V(G) \setminus A}(v) < b$  or  $\deg_B(v) > b$  or adjacent to an  $A$ -vertex  $u$  with  $\deg_A(u) + \deg_U(u) = a$ .

**Rule 8.** Remove from the graph of an instance any edges between  $A$  and  $B$ . Delete the set  $V(H)$  of vertices in any  $b$ -regular component  $H$  of  $G$  such that  $V(H) \subseteq U \cup B$ .

**Rule 9.** Move to  $A$  any  $U$ -vertex  $v$  with  $\deg(v) > b + k$ .

We say that a vertex  $v$  is *tightly-connected* from a  $U$ -vertex  $u$  if there is a path  $P$  from  $u$  to  $v$  such that each vertex  $w \in V(P) \setminus \{u\}$  is a  $U$ -vertex with  $\deg_{V(G) \setminus A}(w) = b$ . For each  $U$ -vertex  $u$ , let  $T(u)$  denote the set  $U$ -vertices tightly-connected from  $u$ , which has the following property: when we include a  $U$ -vertex  $u$  to  $A$ , all the vertices  $T(u)$  need to be included to  $A$ , because the degree of each vertex  $v \in T(u) \setminus \{u\}$  in  $G[U \cup B]$  will be less than  $b$ . Hence if we include a  $U$ -vertex  $u$  with  $|T(u)| > k$ , then  $|A|$  will increase by  $|T(u)| > k$  and the resulting instance cannot have a solution.

**Rule 10.** Move to  $B$  any  $U$ -vertex  $u$  with  $|T(u)| > k$ .

**Rule 11.** Conclude that an instance is infeasible if  $|B \cap N(U)| > bk$  or  $|E(B)| > b(b + 1)k$ .

In what follows, we assume that  $b(b + 1)k > |E(B)| \geq |N(B)|$ . By Rule 10, it holds that  $|T(u)| \leq k$  for each vertex  $u \in N(B)$ . Let  $T^* = N(B) \cup (\cup_{u \in N(B)} T(u))$ . Then  $|T^*| \leq |N(B)|(k + 1) \leq b(b + 1)k(k + 1)$ . We have



**Lemma 5.** *When none of Rule 6-Rule 11 is applicable, it holds that  $|T^*| = O(b^2k^2)$ .*

We compute a maximal set  $\mathcal{S}$  of vertex-disjoint  $(b+1)$ -stars in the induced graph  $G[U]$ . We see that an instance  $I = (G, A, B, a, b)$  is infeasible if  $G[U]$  contains more than  $k$  vertex-disjoint  $(b+1)$ -stars. This is because  $|V_A| \leq k$  means that at least one  $(b+1)$ -star must become disjoint with  $V_A$  and a center  $v$  of the star would satisfy  $\deg_{V_B}(v) \geq b+1$ .

**Rule 12.** *Conclude that an instance is infeasible if  $|\mathcal{S}| > k$ .*

Let  $S_0$  be the set of all vertices in the  $(b+1)$ -stars in  $\mathcal{S}$ . For each integer  $i > 0$ , we denote by  $S_i$  the set  $U \cap N(S_{i-1}) \setminus (T^* \cup (\cup_{j=0}^{i-1} S_j))$ . Let  $S^* = \cup_{i \geq 0} S_i$ .

**Lemma 6.** *When none of Rule 6-Rule 12 is applicable, every  $U$ -vertex  $u$  with  $\deg_U(u) \geq b+1$  is in  $S_0 \cup S_1$ . For each vertex  $v \in U \setminus (T^* \cup S_0 \cup S_1)$ , it holds  $\deg_{V(G) \setminus A}(v) = \deg_U(v) = b$ .*

**Lemma 7.** *When none of Rule 6-Rule 12 is applicable, it holds that  $|S^*| = O((b+1)(b+k)k^2)$ .*

**Lemma 8.** *When none of Rule 6-Rule 12 is applicable, any  $U \setminus (T^* \cup S^*)$ -vertex is in a component  $H$  of  $G[U]$  such that  $V(H) \subseteq U \setminus (T^* \cup S^*)$  and  $V(H) \cap N(A) \neq \emptyset$ .*

We call a component  $H$  of  $G[U]$  *residual* if  $V(H) \subseteq U \setminus (T^* \cup S^*)$  and  $V(H) \cap N(A) \neq \emptyset$ . For a vertex  $u$  in a residual component  $H$ , it holds that  $\deg_{V(G) \setminus A}(u) = \deg_U(u) = b$  for  $u \in V(H) \cap N(A)$ , and  $\deg(u) = \deg_U(u) = b$  for  $u \in V(H) \setminus N(A)$  by Lemma 6.

**Lemma 9.** *Let  $H$  be a residual component in  $G[U]$  of an instance. Then any  $(a, b)$ -regular partition  $(V_A, V_B)$  satisfies either  $V(H) \subseteq V_A$  or  $V(H) \subseteq V_B$ .*

Hence if a residual component  $H$  contains a vertex  $u \in V(H) \cap N(A)$  with  $\deg(u) \neq a$  or is adjacent to an  $A$ -vertex  $v$  with  $\deg_H(v) > a$ , then  $V(H)$  cannot be contained in a set  $V_A$  of any  $(a, b)$ -regular partition  $(V_A, V_B)$ .

**Rule 13.** *Move to  $B$  all vertices in a residual component  $H$  that satisfies one of the following:*

- (i) *There is a vertex  $u \in V(H) \cap N(A)$  with  $\deg(u) \neq a$ ; and*
- (ii) *There is an  $A$ -vertex  $v$  with  $\deg_H(v) > a$ .*

By Lemma 8, we know that each  $U$ -vertex is either in  $T^* \cup S^*$  or a residual component. Note that for any vertex  $u \in V(H) \cap N(A)$  in a residual component  $H$ , it holds  $\deg(u) = \deg_U(u) + \deg_A(u) \geq b+1$ , which indicates that  $\deg(u) \geq b+1 > a$  if  $a \leq b$ . Hence when  $a \leq b$ , after Rule 13 is applied, there is no residual component. We get the following lemma by Lemma 5 and Lemma 7.

**Lemma 10.** *If  $a \leq b$ , then the number  $|U|$  of undecided vertices in the instance after applying all above rules is  $O((b + 1)(b + k)k^2)$ .*

**Lemma 11.** *Assume that there is a residual component  $H$  in  $G[U]$ . Then  $a > b$ ,  $V(H) \subseteq N(A)$ ,  $|V(H)| \leq k$ , and every vertex in  $u \in V(H)$  satisfies  $\deg_U(u) = b$  and  $\deg_A(u) = a - b$ .*

Next we consider the case that  $a > b$ . Let all the vertices in  $A$  be indexed by  $w_1, w_2, \dots, w_{|A|}$ , and define the code  $c(H)$  of a residual component  $H$  in  $G[U]$  to be a vector

$$(\deg_H(w_1), \deg_H(w_2), \dots, \deg_H(w_{|A|})),$$

where  $0 \leq \deg_H(w_i) \leq a$  for each  $i$ . We say that two residual components  $H$  and  $H'$  are equivalent if they have the same code  $c(H) = c(H')$ , where we see that  $|V(H)| = |V(H')|$  since each vertex  $u$  in a residual component has the same degrees in  $A$  and  $U$  by Lemma 11. Hence the feasibility of the instance is independent of the current graph structure among equivalent components. Moreover, if there are more than  $a$  equivalent components, then one of them is not contained in  $V_A$  of some  $(a, b)$ -regular partition when the instance is feasible.

**Rule 14.** *If there are more than  $a$  equivalent residual components for some code, choose arbitrarily one of them and include the vertices of the component to  $B$ .*

**Lemma 12.** *The number of vertices in all residual components is  $O((ak)^{(a-b+1)k})$ .*

By Lemma 5, Lemma 7, and Lemma 12, we have the following.

**Lemma 13.** *If  $a > b$ , the number  $|U|$  of undecided vertices in any instance after applying all above rules is  $O((b + 1)(b + k)k^2 + (ak)^{(a-b+1)k})$ .*

We finally derive an upper bound on the size of  $B$  in an instance  $I$ . Let  $B_1 = B \cap N(U)$  and  $B_2 = B \setminus B_1$ , where  $\deg_B(u) < b$  for each vertex  $u \in B_1$  by Rule 6, and  $\deg_B(u) = b$  for each vertex  $u \in B_2$ . Note that if  $b \leq 1$  then  $B_2 = \emptyset$  by Rule 8, and that if  $|E(B_1, B_2)|$  is odd then  $b$  is also odd since  $b|B_2| - |E(B_1, B_2)| = 2|E(G[B_2])|$ . Observe that the feasibility of  $I$  will not change even if we replace the subgraph  $G[B_2]$  with a smaller graph  $G'$  of degree- $b$   $B$ -vertices as long as each vertex  $u \in B_1$  has the same degree  $\deg_{V(G')}(u) = \deg_{B_2}(u)$  as before. The next lemma ensures that there is such a graph  $G'$  with  $O(|B_1| + b^2)$  vertices.

**Lemma 14.** *Let  $b \geq 2$  be an integer,  $V_1 = \{u_1, u_2, \dots, u_n\}$  be a set of  $n$  vertices, and  $\delta = (d_1, d_2, \dots, d_n)$  be a sequence of nonnegative integers at most  $b - 1$  such that  $b$  is odd if  $d = \sum_{1 \leq i \leq n} d_i$  is odd. Then there is a graph  $G' = (V_2, E_2)$  with  $|V_2| \leq n + b^2 + b + 1$  and a set  $E(V_1, V_2)$  of  $d$  edges between  $V_1$  and  $V_2$  such that after adding  $E(V_1, V_2)$  between  $V_1$  and  $V_2$ , it holds that  $\deg_{V_2}(u_i) = d_i$  for each  $u_i \in V_1$  and  $\deg_{V_1 \cup V_2}(v_i) = b$  for each  $v_i \in V_2$ . Such a pair of graph  $G'$  and edge set  $E(V_1, V_2)$  can be constructed in polynomial time in  $n$ .*

**Rule 15.** When  $b \geq 2$ , remove the subgraph  $G[B_2]$ , and add a graph  $G' = (V_2, E_2)$  with edge set  $E(V_1 = B_1, V_2)$  according to Lemma 14, where  $n = |B_1|$ ,  $V_1 = B_1 = \{u_1, u_2, \dots, u_n\}$  and  $\delta = (\deg_{B_2}(u_1), \deg_{B_2}(u_2), \dots, \deg_{B_2}(u_n))$ .

**Lemma 15.** After applying all above rules, the number of vertices in  $A$  is at most  $k$  and the number of vertices in  $B$  is  $O(bk + b^2)$ .

**Proof.** After Rule 6, the number of vertices in  $A$  is at most  $k$ . After Rule 15, all new vertices added in Rule 15 will form the new vertex set  $B_2$ . Then  $|B| = |B_1| + |B_2| = |B_1| + |V_2| \leq 2|B_1| + b^2 + b + 1 = 2bk + b^2 + b + 1$ .  $\square$

Lemma 10, Lemma 13, and Lemma 15 establish Theorem 3.

## 5 Fixed-Parameter Intractability

This section discusses the fixed-parameter intractability of our problems.

**Theorem 4.** UPPER-DEGREE-BOUNDED BIPARTITION is  $W[2]$ -hard with parameter  $k = |V_A|$ .

For UPPER-DEGREE-BOUNDED BIPARTITION, we give a reduction from DOMINATING SET, a well-known  $W[2]$ -hard problem. DOMINATING SET asks us to test whether a graph  $G$  admits a vertex subset  $D \subseteq V(G)$  of size  $k$  such that each vertex in  $V(G) \setminus D$  is adjacent to at least one vertex in  $D$ . Given an instance  $I = (G, k)$  of DOMINATING SET with a graph  $G$  of maximum degree  $d \geq 2$ , we augment  $G$  to  $G' = (V(G) \cup V_1, E(G) \cup E_1)$  so that each vertex  $v \in V(G)$  will be of degree  $d$  by adding  $d - \deg(v)$  new vertices adjacent to only  $v$ , where  $V_1$  and  $E_1$  are the sets of new added degree-1 vertices and edges, respectively. Let  $I' = (G', a = d, b = d - 1, k)$  be an instance of UPPER-DEGREE-BOUNDED BIPARTITION. We prove that  $I$  is a yes-instance if and only if  $I'$  is feasible. If  $G$  has a dominating set  $D$  of size at most  $k$ , then  $(V_A = D, V(G') \setminus D)$  is a solution to  $I'$ , because each degree- $d$  vertex in  $G'$  is adjacent to at least one vertex in  $D$ , and  $\deg_{V(G') \setminus D}(u) \leq \max\{1, d - 1\}$  holds for each vertex  $u \in V(G') \setminus D$ . When  $I'$  is feasible, we claim that  $I'$  always admits a solution  $(V_A, V_B)$  such that  $V_A \subseteq V(G)$ . The reason is that any vertex  $v \in V_A \setminus V(G)$  must be a degree-1 vertex in  $G'$  whose unique neighbor  $u$  is in  $V(G)$ , and thereby we can replace  $v$  with  $u$  in  $V_A$  to get another solution to  $I'$ . For a solution  $(V_A \subseteq V(G), V_B)$  to  $I'$ , we see that  $V_A$  is a dominating set in the original graph  $G$ .

For REGULAR BIPARTITION, we will show that a special case of this problem is equivalent to PERFECT CODE in  $d$ -regular graphs. PERFECT CODE asks us to test whether  $G$  admits a set  $S \subseteq V(G)$  of at most  $k$  vertices such that for each vertex  $v \in V(G)$  there is precisely one vertex in  $N[v] \cap S$ . It is  $W[1]$ -hard when  $k$  is taken as the parameter [9]. It is easy to see that an instance  $(G, k)$  of PERFECT CODE in a  $d$ -regular graph  $G$  is yes if and only if the instance  $(G, 0, d - 1, k)$  of REGULAR BIPARTITION is feasible. It is quite possible that PERFECT CODE with parameter  $k$  remains  $W[1]$ -hard even if input graphs are restricted to regular graphs.

**Acknowledgments.** We appreciate one of the anonymous reviewers for providing the simple reduction from DOMINATING SET to prove the  $W[2]$ -hardness of UPPER-DEGREE-BOUNDED BIPARTITION in Section 5. The first author is supported by National Natural Science Foundation of China under the Grant 61370071 and Fundamental Research Funds for the Central Universities under the Grant ZYGX2012J069.

## References

1. Amini, O., Saub, I., Saurabh, S.: Parameterized complexity of finding small degree-constrained subgraphs. *Journal of Discrete Algorithms* **10**, 70–83 (2012)
2. Bazgan, C., Tuza, Z., Vanderpooten, D.: Efficient algorithms for decomposing graphs under degree constraints. *Discrete Applied Mathematics* **155**, 979–988 (2007)
3. Bazgan, C., Tuza, Z., Vanderpooten, D.: Degree-constrained decompositions of graphs: bounded treewidth and planarity. *Theoretical Computer Science* **355**(3), 389–395 (2006)
4. Bazgan, Cristina, Tuza, Zsolt, Vanderpooten, Daniel: On the Existence and Determination of Satisfactory Partitions in a Graph. In: Ibaraki, Toshihide, Katoh, Naoki, Ono, Hirotaka (eds.) *ISAAC 2003*. LNCS, vol. 2906, pp. 444–453. Springer, Heidelberg (2003)
5. Betzler, N., Bredereck, R., Niedermeier, R., Uhlmann, J.: On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics* **160**(1–2), 53–60 (2012)
6. Chvátal, V.: Recognizing decomposable graphs. *J. Graph Theory* **8**, 51–53 (1984)
7. Diwan, A.: Decomposing graphs with girth at least five under degree constraints. *J. Graph Theory* **33**, 237–239 (2000)
8. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: On completeness for  $W[1]$ . *Theoretical Computer Science A* **141**(1–2), 109–131 (1995)
9. Fellows, M.R., Guo, J., Moser, H., Niedermeier, R.: A generalization of Nemhauser and Trotter’s local optimization theorem. *Journal of Computer and System Sciences* **77**, 1141–1158 (2011)
10. Grinstead, D.L., Slatara, J., Sherwani, N.A., Holmesc, N.D.: Efficient edge domination problems in graphs. *Information Processing Letters* **48**, 221–228 (1993)
11. Garey, M.R., Johnson, D.S.: *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco (1979)
12. Kaneko, A.: On decomposition of triangle-free graphs under degree constraints. *J. Graph Theory* **27**, 7–9 (1998)
13. Lin, Min Chih, Mizrahi, Michel J., Szwarcfiter, Jayme L.: An  $O^*(1.1939^n)$  time algorithm for minimum weighted dominating induced matching. In: Cai, Leizhen, Cheng, Siu-Wing, Lam, Tak-Wah (eds.) *Algorithms and Computation*. LNCS, vol. 8283, pp. 558–567. Springer, Heidelberg (2013)
14. Mathieson, L., Szeider, S.: The parameterized complexity of regular subgraphs problems and generalizations. In: *CATS 2008, CRPIT 77*, pp. 79–86 (2008)
15. Moser, H., Thilikos, D.: Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms* **7**, 181–190 (2009)
16. Stiebitz, M.: Decomposing graphs under degree constraints. *J. Graph Theory* **23**, 321–324 (1996)

# Faster Existential FO Model Checking on Posets

Jakub Gajarský, Petr Hliněný, Jan Obdržálek<sup>(✉)</sup>, and Sebastian Ordyniak

Faculty of Informatics, Masaryk University, Brno, Czech Republic  
{gajarsky,hlineny,obdrzalek,ordyniak}@fi.muni.cz

**Abstract.** We prove that the model checking problem for the existential fragment of first order (FO) logic on partially ordered sets is fixed-parameter tractable (FPT) with respect to the formula and the width of a poset (the maximum size of an antichain). While there is a long line of research into FO model checking on graphs, the study of this problem on posets has been initiated just recently by Bova, Ganian and Szeider (LICS 2014), who proved that the existential fragment of FO has an FPT algorithm for a poset of fixed width. We improve upon their result in two ways: (1) the runtime of our algorithm is  $O(f(|\phi|, w) \cdot n^2)$  on  $n$ -element posets of width  $w$ , compared to  $O(g(|\phi|) \cdot n^{h(w)})$  of Bova et al., and (2) our proofs are simpler and easier to follow. We complement this result by showing that, under a certain complexity-theoretical assumption, the existential FO model checking problem does not have a polynomial kernel.

## 1 Introduction

The *model checking* problem, asking whether a logical formula holds true on a given input structure, is a fundamental problem of theoretical computer science with applications in many different areas, e.g. algorithm design or formal verification. One way to see why providing efficient algorithms for model checking is important is to note that such algorithms automatically establish efficient solvability of whole classes of problems. For the first-order (FO) logic, the model checking problem is known to be PSPACE-complete when the formula is part of the input, and polynomial time solvable when the formula is fixed in advance.

However, this does not tell the whole story. In the latter scenario we would like to identify the instances where we could do significantly better—in regard to running times—and quantify these gains. Stated in the parlance of parameterized complexity theory, we wish to identify classes of input structures on which we can evaluate every FO formula  $\phi$  in polynomial time  $f(|\phi|) \cdot n^c$ , where  $c$  is a constant independent of the formula. If it is true, we say that FO model checking problem is *fixed-parameter tractable* (FPT) on this class of structures.

---

Research funded by the Czech Science Foundation under grant 14-03501S.

Sebastian Ordyniak: Research funded by Employment of Newly Graduated Doctors of Science for Scientific Excellence (CZ.1.07/2.3.00/30.0009).

Over the past decade this line of research has been very active and led to several important results on (mainly) undirected graphs, which culminated in the recent result of Grohe, Kreutzer and Siebertz [10], stating that FO model checking is fixed-parameter tractable on all nowhere dense classes of graphs.

In contrast, almost nothing is known about the complexity of FO model checking on other finite algebraic structures. Very recently, Bova, Ganian and Szeider [4] initiated the study of the model checking problem for FO and partially ordered sets. Despite similarities between posets and graphs (e.g., in Hasse diagrams), the existing FO model checking results from graphs do not seem to transfer well to posets, perhaps due to lack of usable notions of “locality” and “sparsity” there. This feeling is supported by several negative results in [4], too.

The main result of Bova et al. [4] then is that the model checking problem for the existential fragment of FO (POSET  $\exists$ -FO-MODEL CHECKING) can be solved in time  $f(|\phi|) \cdot n^{g(w)}$ , where  $n$  is the size of a poset and  $w$  its *width*, i.e. the size of its largest antichain. In the language of parameterized complexity, this means that the problem is FPT in the size of the formula, but only XP with respect to the width of the poset. Note that this is not an easy result since, for instance, posets of fixed width can have unbounded clique-width [4].

The proof in [4] goes by first showing that the model checking problem for the existential fragment of FO is equivalent to the embedding problem for posets (which can be thought as analogous to the induced subgraph problem), and then reducing the embedding problem to a suitable family of instances of the homomorphism problem of certain semilattice structures.

While postponing further formal definitions till Section 2, we now state our main result which improves upon the aforementioned result of Bova et al.:

**Theorem 1.** *POSET  $\exists$ -FO-MODEL CHECKING is fixed-parameter tractable in the formula size and the width of an input poset; precisely, solvable in time  $h(|\phi|, w) \cdot O(n^2)$  where  $n$  is the size of a poset and  $w$  its width.*

Our improvement is two-fold; (1) we show that the existential FO model checking problem is fixed-parameter tractable in *both* the size of the formula and the width of the poset, and (2) we give two simpler proofs of this result, one of them completely *self-contained*. Regarding improvement (2), we use the same reduction of existential FO model checking to the embedding problem from [4], but our subsequent solution to embedding is faster and at the same time much more straightforward and easier to follow.

As stated above, we give two different FPT algorithms solving the poset embedding problem (and thus also the existential FO model checking problem). The first algorithm (Section 3) is a natural, and easy to understand, polynomial-time reduction to a CSP (Constraint Satisfaction Problem) instance closed under min polymorphisms, giving us an  $O(n^4)$  dependence of the running time on the size of the poset. The second algorithm (Section 4) has even better, quadratic, time complexity and works by reducing the embedding problem to a restricted variant of the multicoloured clique problem, which is then efficiently solved.

To complement the previous fixed-parameter tractability results, we also investigate possible kernelization of the embedding problem for posets (Section 5).

We show that the embedding problem does not have a polynomial kernel, unless  $\text{coNP} \subseteq \text{NP}/\text{poly}$ , which is thought to be unlikely. This means the embedding problem (and therefore also the existential and full FO model checking problems) cannot be efficiently reduced to an equivalent instance of size polynomial in the parameter.

Full version of the paper, which includes all omitted proofs, is available at [arxiv.org](http://arxiv.org) [9].

## 2 Preliminaries

### 2.1 Posets and Embedding

A *poset*  $\mathcal{P}$  is a pair  $(P, \leq^P)$  where  $P$  is a set and  $\leq^P$  is a reflexive, antisymmetric, and transitive binary relation over  $P$ . The *size* of a poset  $\mathcal{P} = (P, \leq^P)$  is  $\|\mathcal{P}\| := |P|$ . We say that  $p$  *covers*  $p'$  for  $p, p' \in P$ , denoted by  $p' \triangleleft^P p$ , if  $p' \leq^P p$ ,  $p \neq p'$ , and for every  $p''$  with  $p' \leq^P p'' \leq^P p$  it holds that  $p'' \in \{p, p'\}$ . We say that  $p$  and  $p'$  are *incomparable* (in  $\mathcal{P}$ ), denoted  $p \parallel^P p'$  if neither  $p \leq^P p'$  nor  $p' \leq^P p$ . A *chain*  $C$  of  $\mathcal{P}$  is a subset of  $P$  such that  $x \leq^P y$  or  $y \leq^P x$  for every  $x, y \in C$ . An *anti-chain*  $A$  of  $\mathcal{P}$  is a subset of  $P$  such that for all  $x, y \in P$  it is true that  $x \parallel^P y$ . A *chain partition* of  $\mathcal{P}$  is a tuple  $(C_1, \dots, C_k)$  such that  $\{C_1, \dots, C_k\}$  is a partition of  $P$  and for every  $i$  with  $1 \leq i \leq k$  the poset induced by  $C_i$  is a chain of  $\mathcal{P}$ . The *width* of a poset  $\mathcal{P}$ , denoted by  $\text{width}(\mathcal{P})$  is the maximum cardinality of any anti-chain of  $\mathcal{P}$ .

**Proposition 2.1** ([7, Theorem 1]). *Let  $\mathcal{P}$  be a poset. Then in time  $O(\text{width}(\mathcal{P}) \cdot \|\mathcal{P}\|^2)$ , it is possible to compute both  $\text{width}(\mathcal{P}) = w$  and a corresponding chain partition  $(C_1, \dots, C_w)$  of  $\mathcal{P}$ .*

Let  $\mathcal{Q} = (Q, \leq^Q)$  and  $\mathcal{P} = (P, \leq^P)$  be two posets. An *embedding* from  $\mathcal{Q}$  to  $\mathcal{P}$  is an injective function  $e : Q \rightarrow P$  such that  $q \leq^Q q'$  if, and only if,  $f(q) \leq^P f(q')$  for every  $q, q' \in Q$ . The *embedding problem* for posets is thus defined as:

<p><b>EMBEDDING</b></p> <p><b>Input:</b> Two posets <math>\mathcal{Q} = (Q, \leq^Q)</math> and <math>\mathcal{P} = (P, \leq^P)</math>.</p> <p><b>Question:</b> Is there an embedding from <math>\mathcal{Q}</math> into <math>\mathcal{P}</math>?</p>	<p><b>Parameter:</b> <math>\text{width}(\mathcal{P}), \ \mathcal{Q}\ </math></p>
---	--

### 2.2 Constraint Satisfaction Problems

A *constraint satisfaction problem* (CSP)  $I$  is a triple  $\langle V, D, C \rangle$ , where  $V$  is a finite set of variables over a finite set (domain)  $D$ , and  $C$  is a set of constraints. A *constraint*  $c \in C$  consists of a *scope*, denoted by  $V(c)$ , which is an ordered subset of  $V$ , and a relation, denoted by  $R(c)$ , which is a  $|V(c)|$ -ary relation on  $D$ . For a CSP  $I = \langle V, D, C \rangle$  we sometimes denote by  $V(I)$ ,  $D(I)$ , and  $C(I)$ , its set of variables  $V$ , its domain  $D$ , and its set of constraints  $C$ , respectively. A *solution* to a CSP instance  $I$  is a mapping  $\tau : V \rightarrow D$  such that  $\langle \tau[v_1], \dots, \tau[v_{|V(c)|}] \rangle \in R(c)$  for every  $c \in C$  with  $V(c) = \langle v_1, \dots, v_{|V(c)|} \rangle$ .

Given a  $k$ -ary relation  $R$  over some domain  $D$  and a function  $\phi : D^n \rightarrow D$ , we say that  $R$  is *closed under  $\phi$* , if for all collections of  $n$  tuples  $t_1, \dots, t_n$  from  $R$ , the tuple  $\langle \phi(t_1[1], \dots, t_n[1]), \dots, \phi(t_1[k], \dots, t_n[k]) \rangle$  belongs to  $R$ . The function  $\phi$  is also said to be a *polymorphism of  $R$* . We denote by  $\text{Pol}(R)$  the set of all polymorphisms  $\phi$  such that  $R$  is closed under  $\phi$ .

Let  $I = \langle V, D, C \rangle$  be a CSP instance and  $c \in C$ . We write  $\text{Pol}(c)$  for the set  $\text{Pol}(R(c))$  and we write  $\text{Pol}(I)$  for the set  $\bigcap_{c \in C} \text{Pol}(c)$ . We say that  $I$  is closed under a polymorphism  $\phi$  if  $\phi \in \text{Pol}(I)$ .

We will need the following type of polymorphism. A polymorphism  $\phi : D^2 \rightarrow D$  is a *min polymorphism* if there is an ordering of the elements of  $D$  such that for every  $d, d' \in D$ , it holds that  $\phi(d, d') = \phi(d', d) = \min\{d, d'\}$ .

**Proposition 2.2** ([11, Corollary 4.3]). *Any CSP instance  $I$  that is closed under a min polymorphism (that is provided with the input) can be solved in time  $O((ct)^2)$ , where  $c = |C(I)|$  and  $t$  is the maximum cardinality of any constraint relation of  $I$ .*

### 2.3 Parameterized Complexity

Here we introduce the relevant concepts of parameterized complexity theory. For more details, we refer to text books on the topic [6, 8, 12]. An instance of a parameterized problem is a pair  $\langle x, k \rangle$  where  $x$  is the input and  $k$  a parameter. A parameterized problem is *fixed-parameter tractable* if every instance  $\langle x, k \rangle$  can be solved in time  $f(k) \cdot |x|^c$ , where  $f$  is a computable function, and  $c$  is a constant. **FPT** denotes the class of all fixed-parameter tractable problems.

A *kernelization* [1] for a parameterized problem  $\mathcal{A}$  is a polynomial time algorithm that takes an instance  $\langle x, k \rangle$  of  $\mathcal{A}$  and maps it to an equivalent instance  $\langle x', k' \rangle$  of  $\mathcal{A}$  such that both  $|x'|$  and  $k'$  are bounded by some function  $f$  of  $k$ . The output  $\langle x', k' \rangle$  is called a *kernel*. We say that  $\mathcal{A}$  has a *polynomial kernel* if  $f$  is a polynomial. Every fixed-parameter tractable problem admits a kernel, but not necessarily a polynomial kernel [5].

A *polynomial parameter reduction* from a parameterized problem  $\mathcal{A}$  to a parameterized problem  $\mathcal{B}$  is a polynomial time algorithm, which, given an instance  $\langle x, k \rangle$  of  $\mathcal{A}$  produces an instance  $\langle x', k' \rangle$  of  $\mathcal{B}$  such that  $\langle x, k \rangle$  is a YES-instance of  $\mathcal{A}$  if and only if  $\langle x', k' \rangle$  is a YES-instance of  $\mathcal{B}$  and  $k'$  is bounded by some polynomial of  $k$ . The following results show how polynomial parameter reductions can be employed to prove the non-existence of polynomial kernels.

**Proposition 2.3** ([2, Theorem 8]). *Let  $\mathcal{A}$  and  $\mathcal{B}$  be two parameterized problems such that there is a polynomial parameter reduction from  $\mathcal{A}$  to  $\mathcal{B}$ . If  $\mathcal{B}$  has a polynomial kernel, then so has  $\mathcal{A}$ .*

An *OR-composition algorithm* for a parameterized problem  $\mathcal{A}$  maps  $t$  instances  $\langle x_1, k \rangle, \dots, \langle x_t, k \rangle$  of  $\mathcal{A}$  to one instance  $\langle x', k' \rangle$  of  $\mathcal{A}$  such that the algorithm runs in time polynomial in  $\sum_{1 \leq i \leq t} |x_i| + k$ , the parameter  $k'$  is bounded by a polynomial in the parameter  $k$ , and  $\langle x', k' \rangle$  is a YES-instance if and only if there exists  $1 \leq i \leq t$  such that  $\langle x_i, k \rangle$  is a YES-instance.



**Proposition 2.4** ([3, Lemmas 1 and 2]). *If a parameterized problem  $\mathcal{A}$  has an OR-composition algorithm and its unparameterized version is NP-complete, then  $\mathcal{A}$  has no polynomial kernel, unless  $\text{coNP} \subseteq \text{NP/poly}$ .*

### 2.4 Existential First-Order Logic

In this paper we deal with the, well known, relational first-order (FO) logic. Formulas of this logic are built from (a finite set of) variables, relational symbols, logical connectives ( $\wedge, \vee, \neg$ ) and quantifiers ( $\exists, \forall$ ). A sentence is a formula with no free variables. We restrict ourselves to formulas that are in *prefix normal form*. (A first-order formula is in prefix normal form if all quantifiers occur in front of the formula and all negations occur in front of the atoms.) Furthermore an *existential* first-order formula is a first-order formula in prefix normal form that uses only existential quantifiers.

The problem we are interested in is so-called *model checking problem* for the existential FO formulas (and posets), which is formally defined as follows:

<p><b>POSET <math>\exists</math>-FO-MODEL CHECKING</b></p> <p><b>Input:</b> An existential first-order sentence <math>\phi</math> and a poset <math>\mathcal{P} = (P, \leq^P)</math>.</p> <p><b>Question:</b> Is it true <math>\mathcal{P} \models \phi</math>, i.e., is <math>\mathcal{P}</math> a model of <math>\phi</math>?</p>	<p><b>Parameter:</b> <math>\text{width}(\mathcal{P}),  \phi </math></p>
---	---

We remark here that all first-order formulas in this paper are evaluated over posets. In particular, the vocabulary of these formulas consists of only one binary relation  $\leq^P$  and atoms of these formulas can be either equalities between variables ( $x = y$ ) or applications of the predicate  $\leq^P$  ( $x \leq^P y$ ). For a more detailed treatment of the employed setting, we refer the reader to [4].

As shown in [4], the existential FO model checking problem is closely related to the aforementioned *embedding problem* for posets:

**Proposition 2.5** ([4]). *POSET  $\exists$ -FO-MODEL CHECKING is fixed-parameter tractable if and only if so is EMBEDDING. Moreover, there is a polynomial parameter reduction from EMBEDDING to POSET  $\exists$ -FO-MODEL CHECKING.*

*Proof.* The first statement of the proposition follows immediately from [4, Proposition 1]. The second statement of the proposition follows from the proof of [4, Proposition 1] by observing that the obvious reduction from EMBEDDING to POSET  $\exists$ -FO-MODEL CHECKING is polynomial parameter preserving. □

*Remark 1.* Even though [4] does not state the precise runtime and “instance blow-up” for Proposition 2.5, these can be alternatively bounded from above as follows. For an instance  $(\mathcal{P}, \phi)$  where  $\phi \equiv \exists x_1 \dots \exists x_q. \psi(x_1, \dots, x_q)$ , we exhaustively enumerate all posets  $\mathcal{Q}$  on  $Q = \{x_1, \dots, x_q\}$  (modulo equality = on  $Q$ ) such that  $\mathcal{Q} \models \psi$ , and produce a separate instance of EMBEDDING from this particular  $\mathcal{Q}$  into the same  $\mathcal{P}$ . Then  $\mathcal{P} \models \phi$  if and only if at least one of the constructed EMBEDDING instances is YES. The number of produced instances (of  $\mathcal{Q}$ ) is trivially less than the number of all posets on  $q$  elements factorized by equality,  $< 4^{q^2} = 2^{O(|\phi|^2)}$ , and time spent per each one of them in the construction is  $O(|\phi|^2)$ .

### 3 Fixed-Parameter Tractability Proof

In this section we prove the first half of the main result of our paper (Theorem 1) that the existential FO model checking problem for posets is in FPT. By Proposition 2.5, it is enough to consider the embedding problem for that:

**Theorem 3.1.** *Let  $\mathcal{Q} = (Q, \leq^{\mathcal{Q}})$  and  $\mathcal{P} = (P, \leq^{\mathcal{P}})$  be two posets. Then the embedding problem from  $\mathcal{Q}$  into  $\mathcal{P}$  is fixed-parameter tractable, more precisely, it can be solved in time  $O(\text{width}(\mathcal{P})^{|\mathcal{Q}|} \cdot |\mathcal{Q}|^4 \cdot |\mathcal{P}|^4)$ .*

The remainder of this section is devoted to a proof of the above theorem. Let  $w := \text{width}(\mathcal{P})$  for the rest of this section. The algorithm starts by computing a chain partition  $\mathcal{C} = (C_1, \dots, C_w)$  of  $\mathcal{P}$ . This can be done in time  $O(\text{width}(\mathcal{P}) \cdot |\mathcal{P}|^2)$  by Proposition 2.1.

To make the proof clearer, we will, for an embedding, keep track into which chain each element of  $\mathcal{Q}$  is mapped. We say that an embedding  $e$  from  $\mathcal{Q}$  into  $\mathcal{P}$  is *compatible* with a function  $f$  from  $Q$  to  $\{1, \dots, w\}$  if  $e(q) \in C_{f(q)}$  for every  $q \in Q$ . Observe that every embedding  $e$  is trivially compatible with the unique function  $f$ , where  $f(q) = i$  if and only if  $e(q) \in C_i$ . Also note that there are at most  $(\text{width}(\mathcal{P})^{|\mathcal{Q}|})$  such functions  $f$ .

Our algorithm now will do the following: We generate all possible functions  $f$  (as defined in the previous paragraph) and for each such  $f$  we test whether there is an embedding compatible with  $f$ . The following lemma, stating that we can perform such test efficiently, forms the core of our proof.

**Lemma 3.2.** *Let  $f$  be a function from  $Q$  to  $\{1, \dots, w\}$  where  $w = \text{width}(\mathcal{P})$ . Then one can decide in time  $O(|\mathcal{Q}|^4 \cdot |\mathcal{P}|^4)$  whether there is an embedding  $e$  from  $\mathcal{Q}$  to  $\mathcal{P}$  that is compatible with  $f$ .*

*Proof.* We will prove the lemma by reducing the problem (of finding a compatible embedding) in polynomial time to a CSP instance that is closed under a certain min polymorphism and hence can be solved in polynomial time. We start by defining the CSP instance  $I$  for given  $\mathcal{Q}$ ,  $\mathcal{P}$ ,  $f$ , and  $\mathcal{C}$  as above.

$I$  has one variable  $x_q$  for every  $q \in Q$  whose domain are the elements of  $C_{f(q)}$ . Furthermore, for every pair  $q, q'$  of distinct elements of  $Q$ ,  $I$  contains one constraint  $c_{q,q'}$  whose scope is  $(x_q, x_{q'})$  and whose relation  $R(c_{q,q'})$  contains all tuples  $(p, p')$  such that  $p \in C_{f(q)}$ ,  $p' \in C_{f(q')}$ , and simultaneously

1.  $p \leq^{\mathcal{P}} p'$  iff  $q \leq^{\mathcal{Q}} q'$ ,
2.  $p' \leq^{\mathcal{P}} p$ , iff  $q' \leq^{\mathcal{Q}} q$ .

This completes the construction of  $I$ . Observe that a solution  $\tau : V(I) \rightarrow D(I)$  of  $I$  gives rise to an embedding  $e : Q \rightarrow P$  from  $\mathcal{Q}$  to  $\mathcal{P}$  that is compatible with  $f$  by setting  $e(q) = \tau(x_q)$ . Additionally, every embedding  $e : Q \rightarrow P$  from  $\mathcal{Q}$  to  $\mathcal{P}$  that is compatible with  $f$  gives rise to a solution  $\tau : V(I) \rightarrow D(I)$  of  $I$  by setting  $\tau(x_q) = e(q)$ . Hence,  $I$  has a solution if and only if there is an embedding from  $\mathcal{Q}$  to  $\mathcal{P}$  that is compatible with  $f$  and such an embedding can be easily obtained from a solution of  $I$ .

Concerning the runtime,  $I$  can be constructed in time  $O((|Q| \cdot |P|)^2)$ . Since there are less than  $|Q|^2$  constraints and every constraint relation contains  $O(|P|^2)$  pairs, Proposition 2.2 provides a solution to  $I$  in time  $O((|Q|^2 \cdot |P|^2)^2)$ . To finish it is enough to verify that  $I$  is closed under a certain min polymorphism— Lemma 3.3 below.  $\square$

**Lemma 3.3.** *For every  $\mathcal{Q}, \mathcal{P}, f$ , and  $\mathcal{C}$  defined as above, the CSP instance  $I$  is closed under any min polymorphism that is compatible with the partial order  $\leq^P$ .*

*Proof (of Theorem 3.1).* We can generate the chain partition in time  $O(\text{width}(\mathcal{P}) \cdot |P|^2)$ . Then, for each of the  $(\text{width}(\mathcal{P})^{|\mathcal{Q}|})$  functions  $f$  we test the existence of an embedding compatible with  $f$ , which can be done in time  $O(|Q|^4 \cdot |P|^4)$  by Lemma 3.2. This proves our theorem.  $\square$

### 4 Embedding and Multicoloured Clique

In the previous section we have proved that the embedding problem for posets  $\mathcal{Q}$  and  $\mathcal{P}$  is fixed-parameter tractable w.r.t. both  $\text{width}(\mathcal{P})$  and  $\|\mathcal{Q}\|$ , with the running time of  $O(\text{width}(\mathcal{P})^{\|\mathcal{Q}\|} \cdot \|\mathcal{Q}\|^4 \cdot \|\mathcal{P}\|^4)$ . In this section we improve upon this result by giving an alternative self-contained algorithm for EMBEDDING with running time  $O(\text{width}(\mathcal{P})^{\|\mathcal{Q}\|} \cdot \|\mathcal{Q}\|^3 \cdot \|\mathcal{P}\|^2)$ . In combination with Proposition 2.5 (and Remark 1) we thus finish the proof of main Theorem 1.

This new algorithm achieves better efficiency by exploiting some special properties of the problem that are not fully utilized in the previous reduction to CSP. We pay for this improvement by having to work a little bit harder. The core idea is to show that the problem of finding a compatible embedding is reducible (in polynomial time) to a certain restricted variant of MULTICOLOURED CLIQUE.

The MULTICOLOURED CLIQUE problem takes as an input a graph  $G$  together with a proper  $k$ -colouring of the vertices of  $G$ . The question is whether there is a  $k$ -clique in  $G$ , i.e., a clique consisting of exactly one vertex of each colour.

<b>MULTICOLOURED CLIQUE</b>	<b>Parameter:</b> $k$
<b>Input:</b> A graph $G$ with a proper $k$ -colouring of its vertices.	
<b>Question:</b> Is there a clique (set of pairwise adjacent vertices) of size $k$ in $G$ ?	

Consider a chain partition  $(C_1, \dots, C_w)$  of  $\mathcal{P} = (P, \leq^P)$  where  $w = \text{width}(\mathcal{P})$ , and  $\mathcal{Q} = (Q, \leq^Q)$  with  $Q = \{1, \dots, k\}$ . Let  $f : Q \rightarrow \{1, \dots, w\}$ . For an  $f$ -compatible poset embedding instance from  $\mathcal{Q}$  into  $\mathcal{P}$ , we construct a  $k$ -coloured clique instance  $G$  simply as follows. The vertex set is a disjoint union  $V(G) = V_1 \dot{\cup} \dots \dot{\cup} V_k$  of  $k$  colour classes where  $V_i, 1 \leq i \leq k$ , is a copy of  $C_{f(i)}$ . Let  $p \in V_a, q \in V_b$  be copies of  $p' \in C_{f(a)}, q' \in C_{f(b)}$ . Then  $pq \in E(G)$  if and only if  $a \neq b$  and the following hold;  $p' \leq^P q'$  iff  $a \leq^Q b$ , and  $p' \geq^P q'$  iff  $a \geq^Q b$ . We associate each class  $V_i$  of  $G$  with a linear order  $\leq^G$  naturally inherited from the corresponding chain of  $\mathcal{P}$  (we are not going to compare between different classes).

Trivially, constructed  $G$  is a YES-instance of  $k$ -coloured clique if, and only if,  $\mathcal{Q}$  has an  $f$ -compatible embedding into  $\mathcal{P}$ .

**Lemma 4.1.** *Let  $G$  be a graph on  $V(G) = V_1 \dot{\cup} \dots \dot{\cup} V_k$  constructed from a compatible embedding instance as above. Suppose any  $1 \leq a < b \leq k$ .*

- a) *For any  $p \in V_a$ ,  $q_1, q_2, q_3 \in V_b$  such that  $q_1 \leq^G q_2 \leq^G q_3$  it holds; if  $pq_1, pq_3 \in E(G)$  then also  $pq_2 \in E(G)$ .*
- b) *For any  $p_1, p_2 \in V_a$ ,  $q_1, q_2 \in V_b$  such that  $p_1 \leq^G p_2$ ,  $q_1 \leq^G q_2$  it holds; if  $p_1q_2, p_2q_1 \in E(G)$  then also  $p_1q_1, p_2q_2 \in E(G)$ .*

We call a MULTICOLOURED CLIQUE instance  $G$  *interval-monotone* if the colour classes of  $G$  can be given linear order(s) such that both claims a),b) of Lemma 4.1 are satisfied.

**Corollary 4.2.** *Let  $G$  be an interval-monotone (wrt.  $\leq^G$ ) multicoloured clique instance with colour classes  $V_1, \dots, V_k$ . Let  $I \subseteq \{1, \dots, k\}$ . If  $K_1, \dots, K_\ell \subseteq \bigcup_{i \in I} V_i$  are cliques of size  $|I|$ , then also the set*

$$K = \{ \min_{\leq^G} ((K_1 \cup \dots \cup K_\ell) \cap V_i) \mid i \in I \},$$

*called the minimum of  $K_1, \dots, K_\ell$  wrt.  $\leq^G$  and  $I$ , is a clique in  $G$ . The same holds for analogous maximum of  $K_1, \dots, K_\ell$  wrt.  $\leq^G$  and  $I$ .*

For smooth explanation of our algorithm, we introduce the following shorthand notation. Let  $[i, j] = \{i, i + 1, \dots, j\}$ . Let  $V(G) = V_1 \dot{\cup} \dots \dot{\cup} V_k$ . Then  $N_i(v)$  denotes the set of neighbours of  $v \in V(G)$  in  $V_i$ , and moreover,  $N_I(v) := \bigcup_{i \in I} N_i(v)$  and  $N_I(X) := \bigcap_{v \in X} N_I(v)$ . Provided that  $G$  is equipped with linear order(s)  $\leq^G$  on each  $V_i$ ,  $N_i^\uparrow(v)$  denotes the set of all  $w \in V_i$  such that there is  $w' \in N_i(v)$  and  $w' \leq^G w$  (all the vertices which are “above” some neighbour of  $v$  in  $V_i$ ), and this is analogously extended to  $N_I^\uparrow(v)$  and  $N_I^\uparrow(X)$ .

**Algorithm 4.1.** INPUT: An interval-monotone  $k$ -coloured clique instance  $G$ , the colours classes  $V(G) = V_1 \dot{\cup} \dots \dot{\cup} V_k$  and the order  $\leq^G$  on them.

OUTPUT: YES if  $G$  contains a clique of size  $k$ , and NO otherwise.

ALGORITHM: Dynamically compute, for  $i = 2, 3, \dots, k$ , sets  $MinK^i(v)$  and  $MaxK^i(v)$  where  $v \in V_i$ ; such that  $MinK^i(v)$  is the minimum of all the cliques of size  $i$  in  $G$  which are contained in  $\{v\} \cup V_1 \cup \dots \cup V_{i-1}$  (note, these cliques must contain  $v$ ), or  $\emptyset$  if nonexistent, and  $MaxK^i(v)$  is described analogously.

The computation of  $MaxK^i, MinK^i$  using values  $MaxK^2, \dots, MaxK^{i-1}$  and  $MinK^2, \dots, MinK^{i-1}$  is described in the pseudocode below. Note that we have to compute both  $MinK^i$  and  $MaxK^i$  because we compute  $MinK^i$  from previously computed  $MaxK^j, j < i$ , and vice versa.

1. For every  $v \in V_i$ , set  $X := \{v\}$  and repeat:
  - a) For  $j = i - 1, \dots, 1$ , and as long as  $X \neq \emptyset$ , do the following:
    - find the minimum (wrt.  $\leq^G$ ) element  $x \in N_j(X)$  such that  $j = 1$  or  $\emptyset \neq MaxK^j(x) \subseteq N_{[1, j-1]}^\uparrow(X) \cup \{x\}$ . If  $x$  does not exist then  $X := \emptyset$ , and otherwise set  $X := X \cup \{x\}$ . Continue with next  $j$ .
  - b) Set  $MinK^i(v) := X$ .

2. Analogously finish computation of  $MaxK^i(v)$  using previous  $MinK^j(x)$ .
3. Output YES if there is  $v \in V_k$  such that  $MinK^k(v) \neq \emptyset$ , and NO otherwise.

**Theorem 4.4.** *Algorithm 4.1 correctly solves any instance  $G$  of interval-monotone  $k$ -coloured MULTICOLOURED CLIQUE problem, in time  $O(k \cdot |E(G)|)$ .*

*Proof.* It is enough to prove that the value of each  $MinK^i(v)$  and  $MaxK^i(v)$  is computed correctly in the algorithm. Let  $K_{i,v}$  be the minimum of all the cliques of size  $i$  in  $G$  which are contained in  $\{v\} \cup V_1 \cup \dots \cup V_{i-1}$  (well-defined by Corollary 4.2)—the correct value for  $MinK^i(v)$ . Assume that some  $MinK^i(v) = K'_{i,v}$  value is computed wrong, i.e.,  $K_{i,v} \neq K'_{i,v}$ , and that  $i$  is minimal among such wrong values. Clearly,  $i > 2$ .

If  $K'_{i,v} = \emptyset$  then  $K_{i,v} \neq \emptyset = K'_{i,v}$ . Otherwise we observe that, by the choices  $x \in N_j(X)$  in step 1.a),  $K'_{i,v} \neq \emptyset$  is a clique of size  $i$  in  $G$  contained in  $\{v\} \cup V_1 \cup \dots \cup V_{i-1}$ . Consequently,  $K'_{i,v} \neq \emptyset$  implies  $K_{i,v} \neq \emptyset$ , too.

Let  $K''_{i,v} = K'_{i,v}$  if  $K'_{i,v} \neq \emptyset$ , and otherwise let  $K''_{i,v}$  be the last nonempty value of  $X$  in the course of computation of  $MinK^i(v)$  in step 1.a) of the algorithm. Since the tests in step 1.a) of the algorithm always succeed for  $x$  being  $K_{i,v} \cap V_j$  and  $X = K_{i,v} \cap (V_{j+1} \cup \dots \cup V_i)$ , there exists  $j < i$  (and we choose such  $j$  maximum) such that  $\{x\} = K_{i,v} \cap V_j \neq K''_{i,v} \cap V_j = \{x'\}$ . By the same argument, actually,  $x >^G x'$ .

Now, following iteration  $j$  of step 1.a) of the algorithm (which has “wrongly” chosen  $x'$  instead of  $x$ ), let  $K_0 = MaxK^j(x') \cup (K_{i,v} \cap (V_{j+1} \cup \dots \cup V_i))$ . The minimum of  $K_{i,v}$  and  $K_0$  is also a clique of size  $i$ , by the interval-monotone property and Corollary 4.2, contradicting minimality of  $K_{i,v}$  at  $x$ .

In any case, indeed  $K_{i,v} = K'_{i,v}$ .

It remains to analyse the running time. We consider separately every iteration of step 1, each  $v \in V_i$ , for  $i = 2, \dots, k$ . Thanks to the interval-monotone property of  $G$ , we can preprocess the neighbours of  $v$  into subintervals of the classes  $V_1, \dots, V_{i-1}$  with respect to  $\leq^G$ . This is done in time  $O(|N_{[1,i-1]}(v)|)$ . After that, every iteration  $j$  of step 1.a) takes time  $O(|N_j(v)| \cdot k)$ , and so whole step 1 takes time  $O(k \cdot |N_{[1,i-1]}(v)|)$ . Summing this over  $v$  and  $i$  as in the algorithm we arrive right at the estimate  $O(k \cdot |E(G)|)$ . □

**Corollary 4.5.** *EMBEDDING can be solved in time  $O(\text{width}(\mathcal{P})^{|Q|} \cdot |Q|^3 \cdot |P|^2)$ .*

*Proof.* The reduction from embedding to compatible embedding has been shown within Theorem 3.1. By the reduction here,  $|V(G)| = O(|Q| \cdot |P|)$ ,  $|E(G)| = |V(G)|^2$ , and  $k = |Q|$ . The runtime bound thus follows as in Theorem 3.1. □

## 5 Kernelization Lower Bound

Having shown that the EMBEDDING problem is fixed-parameter tractable, it becomes natural to ask whether it also allows for a polynomial kernel. In this section we will show that this unfortunately is not the case, i.e., we show that

EMBEDDING does not have a polynomial kernel unless  $\text{coNP} \subseteq \text{NP}/\text{poly}$ . Consequently, this also excludes a polynomial kernel for the POSET FO-MODEL CHECKING problem, of which EMBEDDING is a special case. (POSET FO-MODEL CHECKING is an extension of POSET  $\exists$ -FO-MODEL CHECKING to the full FO logic.)

We will show our kernelization lower bound for EMBEDDING using the OR-composition technique outlined by Proposition 2.4. Unfortunately, due to the generality of the EMBEDDING problem it turns out to be very tricky to give an OR-composition algorithm directly for the EMBEDDING problem. To overcome this problem, we introduce a restricted version of EMBEDDING, which we call INDEPENDENT EMBEDDING, for which an OR-composition algorithm is much easier to find and whose unparameterized version is still **NP**-complete, as we prove below.

Let  $\mathcal{I}_k = (I_k, \leq^{\mathcal{I}_k})$  be the poset that has  $k$  mutually incomparable chains consisting of three elements each. Then the INDEPENDENT EMBEDDING problem is defined as follows.

<p>INDEPENDENT EMBEDDING</p> <p><b>Input:</b> A poset <math>\mathcal{P} = (P, \leq^P)</math> and a natural number <math>k</math>.</p> <p><b>Question:</b> Is there an embedding from <math>\mathcal{I}_k</math> to <math>\mathcal{P}</math>?</p>	<p><b>Parameter:</b> <math>\text{width}(\mathcal{P}), k</math></p>
--	--

**NP**-completeness of INDEPENDENT EMBEDDING follows straightforwardly from **NP**-completeness of the ordinary independent set problem on graphs. As to an OR-composition algorithm for INDEPENDENT EMBEDDING, the other ingredient in Proposition 2.4, we do roughly as follows: we first align a given collection of instances to the same (maximum) value of the parameter  $k$ , and then we “stack” these instances on top of one another (all elements of a lower instance are “ $\leq^P$ ” than all those of a higher instance), making a combined instance of INDEPENDENT EMBEDDING which is an OR-composition of all the input instances and whose width does not exceed the maximum of their widths.

Here we formulate these two claims without formal proofs, which can be found in the full version of the paper [9].

**Lemma 5.1.** INDEPENDENT EMBEDDING is **NP**-complete.

**Lemma 5.2.** INDEPENDENT EMBEDDING does not have a polynomial kernel unless  $\text{coNP} \subseteq \text{NP}/\text{poly}$ .

We are now ready to summarize the main result of this section:

**Theorem 5.3.** EMBEDDING, POSET  $\exists$ -FO-MODEL CHECKING and POSET FO-MODEL CHECKING have no polynomial kernel unless  $\text{coNP} \subseteq \text{NP}/\text{poly}$ .

## 6 Conclusions

Besides establishing tractability of existential FO model checking on posets of bounded width, the authors of [4] also considered several other poset invariants,

giving (in-)tractability results for existential FO model checking for these variants. This makes, together with our simplification of proof of their main result, the parameterized complexity of the existential FO model checking on posets rather well understood.

The main direction for further research, suggested already in [4], is the parameterized complexity of model checking of full FO logic on restricted classes of posets, especially on posets of bounded width. This problem is challenging, because currently known techniques for establishing tractability of FO model checking are based on locality of FO and cannot be applied easily to posets—transitivity of  $\leq$  causes that, typically, the whole poset is in a small neighbourhood of some element. On the other hand, attempts to evaluate an FO formula on a Hasse diagram (i.e., on the graph of the cover relation of a poset) fail precisely because of locality of FO.

## References

1. Alon, N., Gutin, G., Kim, E., Szeider, S., Yeo, A.: Solving MAX-r-SAT above a tight lower bound. *Algorithmica* **61**(3), 638–655 (2011)
2. Bodlaender, H.L.: Kernelization: New Upper and Lower Bound Techniques. In: Chen, J., Fomin, F.V. (eds.) *IWPEC 2009*. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
3. Bodlaender, H.L., Downey, R., Fellows, M., Hermelin, D.: On problems without polynomial kernels. *J. Comput. System Sci.* **75**(8), 423–434 (2009)
4. Bova, S., Ganian, R., Szeider, S.: Model checking existential logic on partially ordered sets. In: *CSL-LICS 2014 Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, Article No. 21. ACM, New York (2014), <http://dx.doi.org/10.1145/2603088.2603110>, ISBN: 978-1-4503-2886-9
5. Cai, L., Chen, J., Downey, R., Fellows, M.: Advice classes of parameterized tractability. *Ann. Pure Appl. Logic* **84**(1), 119–138 (1997)
6. Downey, R., Fellows, M.: *Parameterized complexity*. Monographs in Computer Science. Springer (1999)
7. Felsner, S., Raghavan, V., Spinrad, J.: Recognition algorithms for orders of small width and graphs of small dilworth number. *Order* **20**(4), 351–364 (2003)
8. Flum, J., Grohe, M.: *Parameterized Complexity Theory*. Springer (2006)
9. Gajarský, J., Hliněný, P., Obdržálek, J., Ordyniak, S.: Faster existential FO model checking on posets. <http://arxiv.org/abs/1409.4433> (2014)
10. Grohe, M., Kreutzer, S., Siebertz, S.: Deciding first-order properties of nowhere dense graphs. In: *STOC 2014*, pp. 89–98. ACM (2014)
11. Jeavons, P., Cohen, D., Gyssens, M.: Closure properties of constraints. *J. ACM* **44**(4), 527–548 (1997)
12. Niedermeier, R.: *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Ser. Math. Appl. OUP (2006)

# Vertex Cover Reconfiguration and Beyond

Amer E. Mouawad<sup>1</sup>(✉), Naomi Nishimura<sup>1</sup>, and Venkatesh Raman<sup>2</sup>

<sup>1</sup> David R. Cheriton School of Computer Science, University of Waterloo,  
Waterloo, ON, Canada

{aabdou, nishi}@uwaterloo.ca

<sup>2</sup> The Institute of Mathematical Sciences, Chennai, India  
vraman@imsc.res.in

**Abstract.** In the VERTEX COVER RECONFIGURATION (VCR) problem, given graph  $G = (V, E)$ , positive integers  $k$  and  $\ell$ , and two vertex covers  $S$  and  $T$  of  $G$  of size at most  $k$ , we determine whether  $S$  can be transformed into  $T$  by a sequence of at most  $\ell$  vertex additions or removals such that each operation results in a vertex cover of size at most  $k$ . Motivated by recent results establishing the  $\mathbf{W}[1]$ -hardness of VCR when parameterized by  $\ell$ , we delineate the complexity of the problem restricted to various graph classes. In particular, we show that VCR remains  $\mathbf{W}[1]$ -hard on bipartite graphs, is  $\mathbf{NP}$ -hard but fixed-parameter tractable on graphs of bounded degree, and is solvable in time polynomial in  $|V(G)|$  on even-hole-free graphs and cactus graphs. We prove  $\mathbf{W}[1]$ -hardness and fixed-parameter tractability via two new problems of independent interest.

## 1 Introduction

Under the reconfiguration framework, we consider structural and algorithmic questions related to the solution space of a search problem  $\mathcal{Q}$ . Given an instance  $\mathcal{I}$ , an optional range  $[r_l, r_u]$  bounding a numerically quantifiable property  $\Psi$  of feasible solutions for  $\mathcal{Q}$ , and a symmetric adjacency relation (usually polynomially-testable)  $\mathcal{A}$  on the set of feasible solutions, we can construct a *reconfiguration graph*  $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$  for each instance  $\mathcal{I}$  of  $\mathcal{Q}$ . The nodes of  $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$  correspond to the feasible solutions of  $\mathcal{Q}$  having  $r_l \leq \Psi \leq r_u$ , with an edge between each pair of nodes corresponding to solutions adjacent under  $\mathcal{A}$  (viewed as a *reconfiguration step* transforming one solution into the other). One can ask if there exists a walk (*reconfiguration sequence*) in  $R_{\mathcal{Q}}(\mathcal{I}, r_l, r_u)$  between feasible solutions  $S$  and  $T$  of  $\mathcal{I}$ , or for the shortest such walk.

These types of reconfiguration questions have received considerable attention in recent literature [9, 11] and are interesting for a variety of reasons. From an algorithmic standpoint, reconfiguration problems model dynamic situations in which we seek to transform a solution into a more desirable one, maintaining feasibility during the process. Moreover, the study of reconfiguration yields insights

---

Amer E. Mouawad and Naomi Nishimura: Research supported by the Natural Science and Engineering Research Council of Canada.



into the structure of the solution space of the underlying problem, crucial for the design of efficient algorithms [3, 9]. Reconfiguration problems have so far been studied mainly under classical complexity assumptions, with most work devoted to determining the existence of a reconfiguration sequence between two given solutions. For most **NP**-complete problems, this question has been shown to be **PSPACE**-complete [11], while for some problems in **P**, the reconfiguration question could be either in **P** [11] or **PSPACE**-complete [2]. As **PSPACE**-completeness implies that the length of reconfiguration sequences can be exponential in the size of the input graph, it is natural to ask whether tractability is possible when the running time depends on the length of the sequence or on other properties of the problem. These results motivated Mouawad et al. [13] to study reconfiguration under the *parameterized complexity* framework [5].

**Overview of Our Results.** For the VERTEX COVER RECONFIGURATION (VCR) problem, the node set of  $R_{VC}(G, 0, k)$  consists of all vertex covers of size at most  $k$  of an  $n$ -vertex graph  $G$  and two nodes are adjacent in  $R_{VC}(G, 0, k)$  if the vertex cover corresponding to one can be obtained from the other by the addition or removal of a single vertex. VCR is known to be **PSPACE**-hard even when restricted to planar graphs of maximum degree three [10]. Recently [13], the problem was shown to be **W[1]**-hard when parameterized by  $\ell$ . Hence, for the general case, one cannot hope for an algorithm solving the problem in  $\mathcal{O}(f(\ell)(nk)^{\mathcal{O}(1)})$  time, for some computable function  $f$ . However, as noted by Mouawad et al. [13], there is a close relation between the fixed-parameter tractability of the problem parameterized by  $\ell$  and the size of the symmetric difference of the two vertex covers in question. In particular, when the size of the symmetric difference is greater than  $\ell$ , we have a trivial no-instance. When the size is equal to  $\ell$ , the problem is solvable by a simple  $\mathcal{O}(\ell!)$  time enumeration algorithm. In fact, it is easy to see that even when the size of the symmetric difference is  $\ell - c$ , for any constant  $c$ , we can solve the problem in  $\mathcal{O}(\ell^{\binom{n}{c}})$  time. In some sense, these observations imply that the problem becomes “harder” as the number of “choices” we have to make “outside” of the symmetric difference increases.

In this work, we embark on a systematic investigation of the (parameterized) complexity of the problem to better understand the relationship between the (fixed-parameter) tractability of the problem and the size and structure of the symmetric difference. In Section 3, we show, via a new problem of independent interest, that VCR parameterized by  $\ell$  remains **W[1]**-hard when restricted to bipartite graphs. The main motivation behind considering bipartite graphs is due to another observation relating the structure of the input graph to the size of the symmetric difference. That is, when the size of the symmetric difference is equal to  $n$ , the input graph is bipartite (Observation 2) and  $\ell$  must be greater than or equal to  $n$ . Thus, even if  $\ell = n$ , the enumeration algorithm mentioned above would run in time exponential in  $n$ . Can we do any better if  $\ell \ll n$  and the input graph is bipartite? Our hardness result answers this question in the negative unless **FPT** = **W[1]**. The result also answers a question left open by

Mouawad et al. [13] by providing the first example of a search problem in  $\mathbf{P}$  whose reconfiguration version is  $\mathbf{W}[1]$ -hard parameterized by  $\ell$ .

Interestingly, excluding odd cycles does not seem to make the VCR problem any easier, whereas excluding (induced or non-induced) even cycles puts the problem in  $\mathbf{P}$ . We prove this result in Section 4. Although at first glance this positive result does not seem to be related to the symmetric difference, we show that we can in fact obtain polynomial-time algorithms whenever the symmetric difference has some “nice” properties. We show that for even-hole-free graphs the symmetric difference is indeed a forest. The structure is slightly more complex for cactus graphs. Moreover, in both cases, the number of vertices we need to consider outside of the symmetric difference is bounded by a constant. We note that a similar polynomial-time algorithm for even-hole-free graphs was also recently, and independently, obtained by Kamiński et al. for solving several variants of the INDEPENDENT SET RECONFIGURATION problem [12].

In Section 5, we start by introducing the notion of nice reconfiguration sequences and show that any reconfiguration sequence can be converted into a nice one. A nice reconfiguration sequence can be split into smaller “pieces” where the added/removed vertices in the first and last piece induce independent sets in the input graph  $G$  and the added/removed vertices in all other pieces induce a biclique in  $G$ . For graphs of degree at most  $d$ , each biclique has at most  $d$  vertices on each side. Given this rather intriguing structure, we believe that nice reconfiguration sequences deserve a more careful study. Using the notion of nice reconfiguration sequences, we show in two steps that VCR is  $\mathbf{NP}$ -hard on 4-regular graphs. In the first step, we prove a general hardness result showing that VCR is at least as hard as the compression variant of the VERTEX COVER (VC) problem. Then, we construct a 4-regular gadget  $W_k$  on  $6k^2$  vertices with the following property: There exist two minimum vertex-disjoint vertex covers  $S$  and  $T$  of  $W_k$ , each of size  $3k^2$ , such that there exists a path of length  $6k^2$  between the nodes corresponding to  $S$  and  $T$  in  $R_{\text{VC}}(W_k, 0, 3k^2 + g(k))$  but no such path exists in  $R_{\text{VC}}(W_k, 0, 3k^2 + g(k) - 1)$ , for some computable function  $g$  and  $k - 2 \leq g(k) \leq k + 3$ . The existence of graphs with properties similar to  $W_k$  has played an important role in determining the complexity of other reconfiguration problems [3, 14]. For instance, the existence of a 3-regular version of  $W_k$ , combined with the fact that reconfiguration is at least as hard as compression, would immediately imply that VCR is  $\mathbf{NP}$ -hard on 3-regular graphs.

Finally, we show that even though  $\mathbf{NP}$ -hard on 4-regular graphs, VCR can be solved in  $\mathcal{O}(f(\ell, d)(nk)^{\mathcal{O}(1)})$  time on graphs of degree at most  $d$ , for some computable function  $f$ . This result answers another open question [13], presenting the first fixed-parameter algorithm for VCR parameterized by  $\ell$ , in this case for graphs of bounded degree. The algorithm is rather technical, as it involves reductions to three intermediary problems, uses a structural decomposition of the input graph, and exploits the properties of nice reconfiguration sequences. However, at a very high level, this result relates to the symmetric difference as follows: In any yes-instance of the VCR problem on graphs of degree at most  $d$ , one can easily bound the size of the symmetric difference and the set of

vertices “not too far away” from it, i.e. the distance is some function of  $\ell$  and  $d$ . However, to guarantee that the capacity constraint  $k$ , i.e. the maximum size of a vertex cover, is never violated, it may be necessary to add/remove vertices which are “far” from (maybe not even connected to) the symmetric difference. Hence, the main technical challenge is to show that finding such vertices can be accomplished “efficiently”.

We believe that the techniques used in both our hardness proofs and positive results can be extended to cover a host of graph deletion problems defined in terms of hereditary graph properties [13]. It also remains to be seen whether our **FPT** result can be extended to a larger class of sparse graphs similar to the work of Fellows et al. on local search [7].

## 2 Preliminaries

For general graph theoretic definitions, we refer the reader to the book of Diestel [4]. Unless otherwise stated, we assume that each graph  $G$  is a simple, undirected graph with vertex set  $V(G)$  and edge set  $E(G)$ , where  $|V(G)| = n$  and  $|E(G)| = m$ . The *open neighborhood* of a vertex  $v$  is denoted by  $N_G(v) = \{u \mid (u, v) \in E(G)\}$  and the *closed neighborhood* by  $N_G[v] = N_G(v) \cup \{v\}$ . For a set of vertices  $S \subseteq V(G)$ , we define  $N_G(S) = \{v \notin S \mid (u, v) \in E(G), u \in S\}$  and  $N_G[S] = N_G(S) \cup S$ . The subgraph of  $G$  induced by  $S$  is denoted by  $G[S]$ , where  $G[S]$  has vertex set  $S$  and edge set  $\{(u, v) \in E(G) \mid u, v \in S\}$ .

Vertices  $s$  and  $t$  (vertex sets  $A$  and  $B$ ) are *separated* if there is no edge  $(s, t)$  (no edge  $(a, b)$  for  $a \in A, b \in B$ ). The *distance* between two vertices  $s$  and  $t$  of  $G$ ,  $dist_G(s, t)$ , is the length of a shortest path in  $G$  from  $s$  to  $t$ . For  $r \geq 0$ , the  *$r$ -neighborhood* of a vertex  $v \in V(G)$  is defined as  $N_G^r[v] = \{u \mid dist_G(u, v) \leq r\}$ . We write  $B(v, r) = N_G^r[v]$  and call it a *ball of radius  $r$  around  $v$* ; for  $A \subseteq V(G)$ ,  $B(A, r) = \bigcup_{v \in A} N_G^r[v]$ . Section 5 makes use of the following:

**Observation 1.** *For any graph  $G$  of degree at most  $d$ ,  $v \in V(G)$ , and  $A \subseteq V(G)$ ,  $|B(v, r)| \leq d^{r+1}$  and  $|B(A, r)| \leq |A|d^{r+1}$ .*

To avoid confusion, we refer to *nodes* in reconfiguration graphs, as distinguished from *vertices* in the input graph. We denote an instance of the VCR problem by  $(G, S, T, k, \ell)$ , where  $G$  is the input graph,  $S$  and  $T$  are the *source* and *target* vertex covers respectively,  $k$  is the *maximum allowed capacity*, and  $\ell$  is an upper bound on the length of the reconfiguration sequence we seek in  $R_{VC}(G, 0, k)$ . By a slight abuse of notation, we use upper case letters to refer to both a node in the reconfiguration graph as well as the corresponding vertex cover. For any node  $S \in V(R_{VC}(G, 0, k))$ , the quantity  $k - |S|$  corresponds to the *available capacity* at  $S$ . We partition  $V(G)$  into the sets  $C_{ST} = S \cap T$  (vertices common to  $S$  and  $T$ ),  $S_R = S \setminus C_{ST}$  (vertices to be removed from  $S$  in the course of reconfiguration),  $T_A = T \setminus C_{ST}$  (vertices to be added to form  $T$ ), and  $O_{ST} = V(G) \setminus (S \cup T) = V(G) \setminus (C_{ST} \cup S_R \cup T_A)$  (all other vertices). A vertex is *touched* in the course of a reconfiguration sequence from  $S$  to  $T$  if  $v$  is either added or removed at least once.

Due to space limitations, most proofs have been removed from the current version of the paper. The affected observations, propositions, lemmas, and theorems have been marked with a star.

**Observation 2. (\*)** *For a graph  $G$  and two vertex covers  $S$  and  $T$  of  $G$ ,  $G[S_R \cup T_A]$  is bipartite, and there is no edge  $(u, v)$  for  $u \in S_R \cup T_A$  and  $v \in O_{ST}$ .*

**Observation 3.** *For a graph  $G$  and vertex covers  $S$  and  $T$  of  $G$ , in any reconfiguration sequence of length at most  $\ell$  from  $S$  to  $T$  a vertex can be touched at most  $\ell - |S_R \cup T_A| + 1$  times, each vertex in  $S_R \cup T_A$  being touched an odd number of times and each other vertex being touched an even number of times.*

Throughout this work, we implicitly consider VCR as a parameterized problem with  $\ell$  as the parameter. The reader is referred to the book of Downey and Fellows [5] for more on parameterized complexity. We sometimes use the modified big-Oh notation  $\mathcal{O}^*$  that suppresses all polynomially bounded factors.

### 3 Bipartite Graphs

For a graph  $G = (V, E)$ , a *crown* is a pair  $(W, H)$  satisfying the following properties: (i)  $W \neq \emptyset$  is an independent set of  $G$ , (ii)  $N_G(W) = H$ , and (iii) there exists a matching in  $G[W \cup H]$  which saturates  $H$  [1]. Crown structures have played a central role in the development of kernelization algorithms for the VC problem [1]. We define a  $(k, d)$ -constrained crown as a crown  $(W, H)$  such that  $|H| \leq k$  and  $|W| - |H| \geq d \geq 0$ . Given a bipartite graph  $G = (A \cup B, E)$  and two positive integers  $k$  and  $d$ , the  $(k, d)$ -BIPARTITE CONSTRAINED CROWN ( $(k, d)$ -BCC) problem asks whether  $G$  has a  $(k, d)$ -constrained crown  $(W, H)$  such that  $W \subseteq A$  and  $H \subseteq B$ .

**Lemma 1.**  *$(k, d)$ -BCC parameterized by  $k + d$  is  $\mathbf{W}[1]$ -hard even when the graph,  $G = (A \cup B, E)$ , is  $C_4$ -free and vertices in  $A$  have degree at most two.*

*Proof.* We give an **FPT** reduction from  $k$ -CLIQUE, known to be  $\mathbf{W}[1]$ -hard, to  $(k, \binom{k}{2})$ -BIPARTITE CONSTRAINED CROWN. For  $(G, k)$  an instance of  $k$ -CLIQUE, we let  $V(G) = \{v_1, \dots, v_n\}$  and  $E(G) = \{e_1, \dots, e_m\}$ .

We first form a bipartite graph  $G' = ((X \cup Z) \cup Y, E_1 \cup E_2)$ , where vertex sets  $X$  and  $Y$  contain one vertex for each vertex in  $V(G)$  and  $Z$  contains one vertex for each edge in  $E(G)$ . More formally, we set  $X = \{x_1, \dots, x_n\}$ ,  $Y = \{y_1, \dots, y_n\}$ , and  $Z = \{z_1, \dots, z_m\}$ . The edges in  $E_1$  join each pair of vertices  $x_i$  and  $y_i$  for  $1 \leq i \leq n$  and the edges in  $E_2$  join each vertex  $z$  in  $Z$  to the two vertices  $y_i$  and  $y_j$  corresponding to the endpoints of the edge in  $E(G)$  to which  $z$  corresponds. Since each edge either joins vertices in  $X$  and  $Y$  or vertices in  $Y$  and  $Z$ , it is not difficult to see that the vertex sets  $X \cup Z$  and  $Y$  form a bipartition.

By our construction,  $G'$  is  $C_4$ -free; vertices in  $X$  have degree 1, and since there are no double edges in  $G$ , i.e. two edges between the same pair of vertices, no pair of vertices in  $Y$  can have more than one common neighbour in  $Z$ .

For  $(G', k, \binom{k}{2})$  an instance of  $(k, \binom{k}{2})$ -BCC,  $A = X \cup Z$ , and  $B = Y$ , we claim that  $G$  has a clique of size  $k$  if and only if  $G'$  has a  $(k, \binom{k}{2})$ -constrained crown  $(W, H)$  such that  $W \subseteq A$  and  $H \subseteq B$ .

If  $G$  has a clique  $K$  of size  $k$ , we set  $H = \{y_i \mid v_i \in V(K)\}$ , namely the vertices in  $Y$  corresponding to the vertices in the clique. To form  $W$ , we choose  $\{x_i \mid v_i \in V(K)\} \cup \{z_i \mid e_i \in E(K)\}$ , that is, the vertices in  $X$  corresponding to the vertices in the clique and the vertices in  $Z$  corresponding to the edges in the clique. Clearly  $H$  is a subset of size  $k$  of  $B$  and  $W$  is a subset of size  $k + \binom{k}{2}$  of  $A$ ; this implies that  $|W| - |H| \geq d = \binom{k}{2}$ , as required. To see why  $N_{G'}(W) = H$ , it suffices to note that every vertex  $x_i \in W$  is connected to exactly one vertex  $y_i \in H$  and every degree-two vertex  $z_i \in W$  corresponds to an edge in  $K$  whose endpoints  $\{v_i, v_j\}$  must have corresponding vertices in  $H$ . Moreover, due to  $E_1$  there is a matching between the vertices of  $H$  and the vertices of  $W$  in  $X$ , and hence a matching in  $G'[W \cup H]$  which saturates  $H$ .

Assuming that  $G'$  has a  $(k, \binom{k}{2})$ -constrained crown  $(W, H)$  such that  $W \subseteq X \cup Z$  and  $H \subseteq Y$ , it suffices to show that  $|H|$  must be equal to  $k$ ,  $|W \cap Z|$  must be equal to  $\binom{k}{2}$ , and hence  $|W \cap X|$  must be equal to  $k$ ; from this we can conclude the vertices in  $\{v_i \mid y_i \in H\}$  form a clique of size  $k$  in  $G$  as  $|W \cap Z| = \binom{k}{2}$ , requiring that edges exist between each pair of vertices in the set  $\{v_i \mid y_i \in H\}$ . Moreover, since  $|W \cap X| = k$  and  $N_{G'}(W) = H$ , a matching that saturates  $H$  can be easily found by picking all edges  $(x_i, y_i)$  for  $y_i \in H$ .

To prove the sizes of  $H$  and  $W$ , we first observe that since  $|H| \leq k$ ,  $N_{G'}(W) = H$ , and each vertex in  $Y$  has exactly one neighbour in  $X$ , we know that  $|W \cap X| \leq |H| \leq k$ . Moreover, since  $|W| = |W \cap X| + |W \cap Z|$  and  $|W| - |H| \geq \binom{k}{2}$ , we know that  $|W \cap Z| = |W| - |W \cap X| \geq \binom{k}{2} + |H| - |W \cap X| \geq \binom{k}{2}$ . If  $|W \cap Z| = \binom{k}{2}$  our proof is complete since, by our construction of  $G'$ ,  $H$  is a set of at most  $k$  vertices in the original graph  $G$  and the subgraph induced by those vertices in  $G$  has  $\binom{k}{2}$  edges. Hence,  $|H|$  must be equal to  $k$ . If instead  $|W \cap Z| > \binom{k}{2}$ , since each vertex of  $Z$  has degree two, the number of neighbours of  $W \cap Z$  in  $Y$  is greater than  $k$ , violating the assumptions that  $N_{G'}(W) = H$  and  $|H| \leq k$ .  $\square$

Combining Lemma 1 with an FPT reduction from  $(k, d)$ -BCC to VCR yields the main theorem of this section:

**Theorem 1 (\*)**. *VCR parameterized by  $\ell$  is  $\mathbf{W}[1]$ -hard on bipartite graphs.*

## 4 Even-Hole-Free and Cactus Graphs

A *cactus graph* is a connected graph in which each edge is in at most one cycle. A graph  $G$  is *even-hole-free* if no induced subgraph of  $G$  is a cycle on an even number of vertices. Examples of even-hole-free graphs include trees, interval graphs, and chordal graphs.

We present a characterization of instances of the VCR problem solvable in time polynomial in  $n$ , and then apply this characterization to trees, even-hole-free graphs, and cactus graphs. In all cases, we find reconfiguration sequences of

shortest possible length and therefore ignore the parameter  $\ell$ . Reconfiguration sequences are represented as ordered sequences of nodes in  $R_{VC}(G, 0, k)$ .

**Definition 1.** *Given two vertex covers  $A$  and  $B$  of  $G$ , a reconfiguration sequence  $\beta$  from  $A$  to some vertex cover  $A'$  is a  $c$ -bounded prefix of a reconfiguration sequence  $\alpha$  from  $A$  to  $B$ , denoted  $A \xrightarrow{c, B} A'$ , if and only if all of the following conditions hold: (1)  $|A'| \leq |A|$ , (2) for each node  $A''$  in  $\beta$ ,  $|A''| \leq |A| + c$ , (3) for each node  $A''$  in  $\beta$ ,  $A''$  is obtained from its predecessor by either the removal or the addition of a single vertex in the symmetric difference of the predecessor and  $B$ , and (4) no vertex is touched more than once in the course of  $\beta$ . Moreover,  $A \xrightarrow{c, B} A'$  implies  $A \xrightarrow{d, B} A'$  for all  $d > c$ .*

**Lemma 2 (\*)**. *Given two vertex covers  $S$  and  $T$  of  $G$  and two positive integers  $k$  and  $c$  such that  $|S|, |T| \leq k$ , a reconfiguration sequence  $\alpha$  of length  $|S_R| + |T_A| = |S\Delta T|$  from  $S$  to  $T$  exists if: (1)  $|S| \leq k - c$ , (2)  $|T| \leq k - c$ , and (3) for any two vertex covers  $A$  and  $B$  of  $G$  such that  $|A| \leq k - c$  and  $|B| \leq k - c$ , either  $A \xrightarrow{c, B} A'$  or  $B \xrightarrow{c, A} B'$ , where  $A'$  and  $B'$  are vertex covers of  $G$ . Moreover, if  $c$ -bounded prefixes can be found in time polynomial in  $n$ , then so can  $\alpha$ .*

The proof of Theorem 2 shows that Lemma 2 applies if  $G$  is a tree and  $S$  and  $T$  are of size at most  $k - 1$ , as we can always find 1-bounded prefixes  $S \xrightarrow{1, T} S'$  or  $T \xrightarrow{1, S} T'$  in time polynomial in  $n$ . Further refinements are required when at least one of  $S$  and  $T$  is of size greater than  $k - 1$ .

**Theorem 2 (\*)**. *VCR on trees can be solved in time polynomial in  $n$ .*

The proof of Theorem 2 uses  $G$  being a tree only to establish the fact that  $G[S_R \cup T_A]$  is a forest. This fact holds for any even-hole-free graph, since a graph that is bipartite (Observation 2) and has no induced even cycles is a forest, hence:

**Corollary 1**. *VCR on even-hole-free graphs can be solved in time polynomial in  $n$ .*

To extend Corollary 1 to all cactus graphs (which are not necessarily even-hole-free), we show in Lemmas 3 and 4 that the third condition of Lemma 2 is satisfied for cactus graphs with  $c = 2$  and that 2-bounded prefixes can be found in time polynomial in  $n$ .

**Lemma 3 (\*)**. *Given two vertex covers  $S$  and  $T$  of  $G$ , there exists a vertex cover  $S'$  (or  $T'$ ) of  $G$  such that  $S \xrightarrow{2, T} S'$  (or  $T \xrightarrow{2, S} T'$ ) if one of the following conditions holds: (1)  $G[S_R \cup T_A]$  has a vertex  $v \in S_R$  ( $v \in T_A$ ) such that  $|N_{G[S_R \cup T_A]}(v)| \leq 1$ , or (2) there exists a cycle  $Y$  in  $G[S_R \cup T_A]$  such that all vertices in  $Y \cap S_R$  ( $Y \cap T_A$ ) have degree exactly two in  $G[S_R \cup T_A]$ . Moreover, both conditions can be checked in time polynomial in  $n$  and when one of them is true the corresponding 2-bounded prefix can be found in time polynomial in  $n$ .*

**Lemma 4 (\*)**. *If  $G$  is a cactus graph and  $S$  and  $T$  are two vertex covers of  $G$ , then there exists a vertex cover  $S'$  (or  $T'$ ) of  $G$  such that  $S \xleftrightarrow{2, T} S'$  (or  $T \xleftrightarrow{2, S} T'$ ). Moreover, finding such 2-bounded prefixes can be accomplished in time polynomial in  $n$ .*

**Theorem 3 (\*)**. *VCR on cactus graphs can be solved in time polynomial in  $n$ .*

## 5 Graphs of Bounded Degree

**Nice Edit Sequences.** We represent a partially specified reconfiguration sequence in terms of markers  $\mathcal{E} = \{\emptyset, a, r\} \cup \mathcal{E}_a \cup \mathcal{E}_r$ , where  $\mathcal{E}_a = \{a_1, \dots, a_n\}$  and  $\mathcal{E}_r = \{r_1, \dots, r_n\}$ . An edit sequence  $\alpha$  is an ordered sequence of elements of  $\mathcal{E}$ , where the markers  $a_i, a, r_j, r$ , and  $\emptyset$  represent the addition of vertex  $v_i$ , an unspecified addition, the removal of vertex  $v_j$ , an unspecified removal, and a blank placeholder, respectively. We refer to  $\mathcal{E}_a \cup \{a\}$  and  $\mathcal{E}_r \cup \{r\}$  as the sets of *addition markers* and *removal markers*, respectively. An edit sequence  $\alpha$  is *unlabeled* if it contains no markers in  $\mathcal{E}_a \cup \mathcal{E}_r$ , *partly labeled* if it contains at least one element from each of the sets  $\{a, r\}$  and  $\mathcal{E}_a \cup \mathcal{E}_r$ , and *labeled* if it contains no markers in  $\{a, r\}$ . We say  $\alpha$  is *partial* if it contains at least one  $\emptyset$  and is *full* otherwise.

**Observation 4.** *The total number of possible full (partial) unlabeled edit sequences of length at most  $\ell$  is  $\sum_{i=1}^{\ell} 2^i < 2^{\ell+1}$  ( $\sum_{i=1}^{\ell} 3^i < 3^{\ell+1}$ ).*

The *length* of  $\alpha$ ,  $|\alpha|$ , is the number of markers in  $\alpha$ . We use  $\alpha[p] \in \mathcal{E}$ ,  $1 \leq p \leq |\alpha|$  to denote the marker at position  $p$  in  $\alpha$ , and refer to it as a *blank marker* if  $\alpha[p] = \emptyset$ . By extension,  $\alpha[p_1, p_2]$ ,  $1 \leq p_1 \leq p_2 \leq |\alpha|$ , denotes the edit sequence of length  $p_2 - p_1 + 1$  formed from  $\alpha[p_1]$  through  $\alpha[p_2]$ ; such a sequence is a *segment* of  $\alpha$ . Two segments  $\beta$  and  $\beta'$  are *consecutive* if  $\beta = \alpha[p_1, p_2]$  and  $\beta' = \alpha[p_2 + 1, p_3]$  for some  $p_1 \leq p_2 \leq p_3$ ;  $\beta'$  ( $\beta$ ) is the *successor* (*predecessor*) of  $\beta$  ( $\beta'$ ). A segment  $\beta$  of  $\alpha$  is an *add-remove segment* if  $\beta$  contains addition markers followed by removal markers, and a *d-add-remove segment*,  $d > 0$ , if it is an add-remove segment with  $i$  addition and  $j$  removal markers,  $1 \leq i \leq d$  and  $1 \leq j \leq d$ . A *piece* is a group of zero or more consecutive segments.

**Definition 2.** *Given a positive integer  $d > 0$ , an edit sequence  $\alpha$  is  $d$ -well-formed if it is subdivided into three consecutive pieces such that: (1) The starting piece consists of zero or more removal markers, (2) the central piece consists of zero or more  $d$ -add-remove segments, and (3) the ending piece consists of zero or more addition markers.*

We can form the length  $|\beta| + |\gamma|$  *concatenation*  $\text{concat}(\beta, \gamma)$  of two edit sequences  $\beta$  and  $\gamma$  in the obvious way, and *cut* the marker at position  $p$  by forming  $\text{concat}(\beta[1, p - 1], \beta[p + 1, |\alpha|])$ . The edit sequence  $\text{clean}(\beta)$  is formed by cutting all blank markers. Given a partial edit sequence  $\beta$  and a full edit sequence  $\gamma$ , the *merging* operation consists of replacing the  $p$ th blank marker in  $\beta$  with the

$p$ th marker in  $\gamma$ . We say a full edit sequence  $\gamma$  is a *filling edit sequence* of partial edit sequence  $\beta$  if  $\text{merge}(\beta, \gamma)$  produces a full edit sequence. Given  $t \geq 2$  full labeled edit sequences  $\alpha_1, \dots, \alpha_t$ , the *mixing* of those sequences,  $\text{mix}(\alpha_1, \dots, \alpha_t)$ , produces the set of all full labeled edit sequences of length  $|\alpha_1| + \dots + |\alpha_t|$ . Each  $\alpha \in \text{mix}(\alpha_1, \dots, \alpha_t)$  consists of all markers in each  $\alpha_i$ ,  $1 \leq i \leq t$ , such that the respective orderings of markers from each  $\alpha_i$  is maintained, i.e. if we cut from  $\alpha$  the markers of all sequences except  $\alpha_1$ , we obtain  $\alpha_1$ .

To relate edit and reconfiguration sequences, for graph  $G$  and edit sequence  $\alpha$ , we use  $V(\alpha)$  to denote the set of vertices touched in  $\alpha$ , i.e.  $V(\alpha) = \{v_i \mid a_i \in \alpha \vee r_i \in \alpha\}$ . For  $\alpha$  full and labeled,  $V(S, \alpha)$  denotes the set of vertices obtained after executing all reconfiguration steps in  $\alpha$  on  $G$  starting from some vertex cover  $S$  of  $G$ . If each set  $V(S, \alpha[1, p])$ ,  $1 \leq p \leq |\alpha|$ , is a vertex cover of  $G$ , then  $\alpha$  is *valid* (and *invalid* otherwise). Even if  $|S| \leq k$ ,  $\alpha$  is not necessarily a walk in  $R_{VC}(G, 0, k)$ , as  $\alpha$  might violate the maximum allowed capacity constraint  $k$ . We say  $\alpha$  is *tight* if it is valid and  $\max_{1 \leq p \leq |\alpha|} (|V(S, \alpha[1, p])|) \leq k$ . A partial labeled edit sequence  $\alpha$  is valid or tight if  $\text{clean}(\alpha)$  is valid or tight.

**Observation 5.** *Given a graph  $G$  and two vertex covers  $S$  and  $T$  of  $G$ , an edit sequence  $\alpha$  is a reconfiguration sequence from  $S$  to  $T$  if and only if  $\alpha$  is a tight edit sequence from  $S$  to  $T$ .*

Given a graph  $G$ , a vertex cover  $S$  of  $G$ , a full unlabeled edit sequence  $\alpha$ , and an ordered sequence  $L = \{l_1, \dots, l_{|\alpha|}\}$  of (not necessarily distinct) labels between 1 and  $n$ , the *label* operation, denoted by  $\text{label}(\alpha, L)$ , returns a full labeled edit sequence  $\alpha'$ ; each marker in  $\alpha'$  is copied from  $\alpha$  and assigned the corresponding label from  $L$ . A full unlabeled edit sequence  $\alpha$  can be *applied* to  $G$  and  $S$  if there exists an  $L$  such that  $\text{label}(\alpha, L)$  is valid starting from  $S$ .

**Definition 3.** *For  $t \geq 2$ , graph  $G$ , and vertex cover  $S$  of  $G$ , valid labeled edit sequences  $\alpha_1, \dots, \alpha_t$  are compatible if each  $\alpha \in \text{mix}(\alpha_1, \dots, \alpha_t)$  is a valid edit sequence starting from  $S$ , and incompatible otherwise.*

**Definition 4.** *For a graph  $G$  of degree at most  $d$  and a vertex cover  $S$  of  $G$ , a valid edit sequence  $\alpha$  starting from  $S$  is a nice edit sequence if it is valid,  $d$ -well-formed, and satisfies the following invariants:*

- **Connectivity invariant:**  $G[V(\beta_i)]$  is connected for all  $i$ , where  $\beta_i$  denotes the  $i^{\text{th}}$   $d$ -add-remove segment in the central piece of  $\alpha$ .
- **Early removal invariant:** For  $1 \leq p_1 < p_2 < p_3 \leq |\alpha|$ , if  $\alpha[p_1] \in \mathcal{E}_a$ ,  $\alpha[p_2] \in \mathcal{E}_a$ , and  $\alpha[p_3] \in \mathcal{E}_r$ , then  $V(\alpha[p_3])$  and  $V(\alpha[p_1 + 1, p_3 - 1])$  are not separated.

Intuitively, the early removal invariant states that every removal marker in a nice edit sequence must occur “as early as possible”. In other words, a vertex is removed right after its neighbors (and possibly itself) are added.

**Lemma 5 (\*).** *Given a graph  $G$  of degree at most  $d$  and two vertex covers  $S$  and  $T$  of  $G$ , it is possible to transform any valid edit sequence  $\alpha$  from  $S$  to  $T$  into a nice edit sequence  $\alpha'$  in  $\mathcal{O}(n^4|\alpha|^42^d)$  time such that  $|V(S, \alpha'[1, p])| \leq |V(S, \alpha[1, p])|$  for all  $1 \leq p \leq |\alpha|$ . In other words, if  $\alpha$  is tight then so is  $\alpha'$ .*



**NP-Hardness on 4-Regular Graphs.** We prove our result by demonstrating a reduction from VERTEX COVER COMPRESSION (VCC) to VCR where the input graph is restricted to be 4-regular; given a graph  $G$  and a vertex cover of size  $k$ , VCC asks whether  $G$  has a vertex cover of size  $k - 1$ .

**Theorem 4 (\*).** *VCR is at least as hard as VCC.*

Theorem 4 relies on a reduction involving the disjoint union of an instance of VCC and a biclique  $K_{k,k}$ ; the instance of VCR can be reconfigured only if compression is possible. Using this idea, we show that VCR remains NP-hard for 4-regular graphs by constructing a 4-regular gadget  $W_k$  which will replace the  $K_{k,k}$  biclique. Theorem 5 then follows from the facts that VC is NP-hard on 4-regular graphs [8] and any algorithm which solves the VCC problem can be used to solve VC.

**Theorem 5 (\*).** *VCR is NP-hard on 4-regular graphs.*

**FPT Algorithm for Graphs of Bounded Degree.** We make use of three different problems. In the ANNOTATED VCR (AVCR) problem, the vertex set of the input graph is partitioned into sets  $X$ ,  $W$ , and  $R$  such that  $X$  and  $R$  are separated,  $S_R \cup T_A \subseteq X$ , and we seek a reconfiguration sequence in which no vertex in  $W$  is touched. In the VERTEX COVER WALK (VCW) problem, given a graph  $G$ , a vertex cover  $S$  of  $G$ , and a full unlabeled edit sequence  $\sigma$  of length  $\ell \geq 1$ , the goal is to determine whether we can apply  $\sigma$  to  $G$  and  $S$ . In the parameterized setting, VCW is at least as hard as the problem of determining, given a graph  $G$ , a vertex cover  $S$  of  $G$ , and integer  $\ell \geq 1$ , whether  $G$  has a vertex cover  $S'$  such that  $|S'| < |S|$  and  $|S' \Delta S| \leq \ell$  (VERTEX COVER LOCAL SEARCH (VCLS)) [7], known to be **W[1]**-hard on graphs of bounded degeneracy and **FPT** on graphs of bounded degree [7].

**Lemma 6 (\*).** *When parameterized by  $\ell$ , VCW is at least as hard as VCLS.*

Finally, we also make use of  $\ell$ -MULTICOLORED INDEPENDENT SET ( $\ell$ -MIS), the problem of determining, for a graph  $G$ , a positive integer  $\ell$ , and a (not necessarily proper) vertex-coloring  $c : V(G) \rightarrow \{c_1, \dots, c_\ell\}$ , whether  $G$  has an independent set of size  $\ell$  including exactly one vertex of each color. Using a reduction from the **W[1]**-hard  $\ell$ -MULTICOLORED CLIQUE problem [6] in which we complement all edges in the input graph,  $\ell$ -MIS is **W[1]**-hard in general graphs. For  $c(v)$  the color assigned to  $v \in V(G)$ , we say  $v$  belongs to *color class*  $c(v)$ , and let  $V_i(G)$  denote the set of vertices assigned colored  $c_i$  in  $G$ , i.e.  $V_i(G) = \{v \in V(G) \mid c(v) = c_i\}$ .

**Lemma 7 (\*).** *The  $\ell$ -MIS problem parameterized by  $\ell$  can be solved in **(FPT)**  $\mathcal{O}^*((d\ell)^{2\ell})$  time if for every vertex  $v \in V(G)$  such that  $c(v) = c_i$ ,  $|N_G(v) \cap V_j(G)| \leq d$ , for some fixed constant  $d$ ,  $i \neq j$ , and  $1 \leq i, j \leq \ell$ .*

Our **FPT** algorithm for VCR relies on a combination of enumeration and reductions to the aforementioned problems, starting with a reduction to AVCR:

**Lemma 8 (\*)**. *For any instance of VCR, there exists a set of  $2\ell$  instances  $\{\mathcal{I}_1, \dots, \mathcal{I}_{2\ell}\}$  of AVCR such that the original instance is a yes-instance for VCR if and only if at least one  $\mathcal{I}_x$  is a yes-instance for AVCR,  $1 \leq x \leq 2\ell$  and for each  $X_x, S_R \cup T_A \subseteq X_x$ .*

The algorithm implicit in the proof of Lemma 8 generates  $2\ell$  instances of AVCR such that the original instance is a yes-instance of VCR if and only if a generated instance is a yes-instance of AVCR. In each instance, there is a subset  $W$  of  $C_{ST}$  separating a superset  $X$  of  $S_R \cup T_A$  from a vertex set  $R$  such that no vertex in  $W$  is touched during reconfiguration. We use enumeration to generate all partial labeled edit sequences that touch only vertices in  $X$ ; if any produces a tight sequence that transforms  $S$  to  $T$ , we have a yes-instance. Otherwise, we consider sequences which are valid and transform  $S$  to  $T$  but exceed the capacity constraint. By finding an appropriate labeled filling sequence  $\gamma'$  that touches vertices in  $R$ , we can free up capacity so that  $merge(\beta, \gamma')$  is tight.

We can find such a  $\gamma'$  trivially if there is a sufficiently large independent set in the vertices of  $S \cap R$  with no neighbours in  $O_{ST}$ , as our reconfiguration sequence will consist of removing the vertices in the independent set to free up capacity, applying  $\beta$ , and then adding back the vertices in the independent set. Otherwise, we reduce the problem of finding  $\gamma'$  to an instance of VCW on  $G[R]$  for each suitable unlabeled edit sequence  $\gamma$  of length the number of blanks in  $\beta$ .

**Lemma 9 (\*)**. *If VCW is solvable in  $\mathcal{O}^*(f(d, \ell))$  time, for some computable function  $f$ , on a graph  $G$  of degree at most  $d$ , then VCR is solvable in  $\mathcal{O}^*(2\ell 3^{\ell+1} (\ell d^{2\ell+1})^{2\ell+2} (d + \ell)^\ell 2^\ell f(d, \ell))$  time on  $G$ .*

To try all possible ways of generating  $\gamma'$ , we start by enumerating all  $d$ -well-formed full unlabeled edit sequences  $\gamma$  of the appropriate length, and for each try all possible choices for the starting piece; by Lemma 5 this is sufficient. Given  $\gamma$  and a starting piece, we create  $t$  instances of VCW, where instance  $\mathcal{J}_y$  corresponds to the graph induced by the labeled central piece having  $y$  connected components. To solve instance  $\mathcal{J}_y$ , we consider all ways of assigning the  $d$ -add-remove segments to connected components. This allows us to create a sequence for each component, where  $\gamma_h$  touches only vertices in component  $h$ , and all vertices touched by  $\gamma'_h$  can be found in a ball of radius  $|\gamma'_h|$ .

We can reduce each such subproblem to an instance of  $y$ -MIS satisfying Lemma 7, where a color  $c_h$  corresponds to component  $h$ . We denote the corresponding auxiliary graph by  $G_A$ . In  $G_A$ , we create vertices for each labeled full edit sequence  $\lambda$ , where  $G[V(\lambda)]$  is connected and  $\lambda$  can be derived by adding labels to  $\gamma_h$ . There is an edge between two vertices in  $G_A$  if they have different colors and the sets of vertices associated with their edit sequences are not separated. Thus, a solution to  $y$ -MIS indicates that there are  $y$  full labeled edit sequences with separated vertex sets, as required to complete the central piece of  $\gamma$ . As the ending piece of  $\gamma'$  will be determined by the starting and central pieces, this completes the algorithm.

**Lemma 10 (\*)**. *Each instance of the VCW problem generated by our algorithm for AVCR can be solved in  $\mathcal{O}^*(f(d, \ell))$  time, for some computable function  $f$ , on graphs of degree at most  $d$ .*

Combining Lemmas 9 and 10 yields the main theorem of this section:

**Theorem 6**. *For every fixed constant  $d$ , VCR parameterized by  $\ell$  is fixed-parameter tractable for graphs of degree at most  $d$ .*

## References

1. Abu-Khazam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: Theory and experiments. In: Proc. of the Sixth Workshop on Algorithm Engineering and Experiments, pp. 62–69 (2004)
2. Bonsma, P.: The complexity of rerouting shortest paths. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 222–233. Springer, Heidelberg (2012)
3. Cereceda, L., van den Heuvel, J., Johnson, M.: Connectedness of the graph of vertex-colourings. *Discrete Mathematics* **308**(56), 913–919 (2008)
4. Diestel, R.: *Graph Theory*. Electronic Edition. Springer (2005)
5. Downey, R.G., Fellows, M.R.: *Parameterized complexity*. Springer, New York (1997)
6. Fellows, M.R., Hermelin, D., Rosamond, F., Vialette, S.: On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.* **410**(1), 53–61 (2009)
7. Fellows, M.R., Rosamond, F.A., Fomin, F.V., Lokshtanov, D., Saurabh, S., Villanger, Y.: Local search: is brute-force avoidable? In: Proc. of the 21st International Joint Conference on Artificial Intelligence, pp. 486–491 (2009)
8. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. *Theor. Comput. Sci.* **1**(3), 237–267 (1976)
9. Gopalan, P., Kolaitis, P.G., Maneva, E.N., Papadimitriou, C.H.: The connectivity of boolean satisfiability: computational and structural dichotomies. *SIAM J. Comput.* **38**(6), 2330–2355 (2009)
10. Hearn, R.A., Demaine, E.D.: PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.* **343**(1–2), 72–96 (2005)
11. Ito, T., Demaine, E.D., Harvey, N.J.A., Papadimitriou, C.H., Sideri, M., Uehara, R., Uno, Y.: On the complexity of reconfiguration problems. *Theor. Comput. Sci.* **412**(12–14), 1054–1065 (2011)
12. Kamiński, M., Medvedev, P., Milanič, M.: Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.* **439**, 9–15 (2012)
13. Mouawad, A.E., Nishimura, N., Raman, V., Simjour, N., Suzuki, A.: On the parameterized complexity of reconfiguration problems. In: Gutin, G., Szeider, S. (eds.) IPEC 2013. LNCS, vol. 8246, pp. 281–294. Springer, Heidelberg (2013)
14. Suzuki, A., Mouawad, A.E., Nishimura, N.: Reconfiguration of dominating sets. In: Cai, Z., Zelikovskiy, A., Bourgeois, A. (eds.) COCOON 2014. LNCS, vol. 8591, pp. 405–416. Springer, Heidelberg (2014)

# **Graph Algorithms: Approximation I**

# Approximating the Maximum Internal Spanning Tree Problem via a Maximum Path-Cycle Cover

Xingfu Li and Daming Zhu<sup>(✉)</sup>

School of Computer Science and Technology, Shandong University,  
Jinan 250101, China  
lxf@mail.sdu.edu.cn, dmzhu@sdu.edu.cn

**Abstract.** This paper focuses on finding a spanning tree of a graph to maximize its internal vertices in number. We propose a new upper bound for the number of internal vertices in a spanning tree, which shows that for any undirected simple graph, any spanning tree has less internal vertices than the edges a maximum path-cycle cover has. Thus starting with a maximum path-cycle cover, we can devise an approximation algorithm with a performance ratio  $\frac{3}{2}$  for this problem on undirected simple graphs. This improves upon the best known performance ratio  $\frac{5}{3}$  achieved by the algorithm of Knauer and Spoerhase. Furthermore, we can improve the algorithm to achieve a performance ratio  $\frac{4}{3}$  for this problem on graphs without leaves.

**Keywords:** Algorithm · Complexity · Approximation · Maximum Internal Spanning Tree

## 1 Introduction

The *Maximum Internal Spanning Tree* problem, MIST briefly, is motivated by the design of cost-efficient communication networks[14]. It asks to find a spanning tree of a graph such that its number of internal vertices is maximized. Since a Hamilton path (if exists) of a graph is also a spanning tree of that graph with its internal vertices maximized, and finding a Hamilton path in a graph is NP-Hard classically[7], MIST is NP-hard of course.

MIST admits approximation algorithms with a constant performance ratio. Prieto et al. [11] first presented a 2-approximation local search algorithm in 2003. Later, by a modification of depth-first search, Salamon et al [14] improved Prieto's 2-approximation algorithm to running in linear-time. Besides, they proposed a  $\frac{3}{2}$ -approximation algorithm on claw-free graphs and a  $\frac{6}{5}$ -approximation algorithm on cubic graphs [14]. Salamon even showed that his 2-approximation algorithm in [14] can achieve a performance ratio  $\frac{r+1}{3}$  on  $r$ -regular graphs [16]. Furthermore, by local optimization, Salamon [15] devised a  $O(n^4)$ -time and  $\frac{7}{4}$ -approximation algorithm on graphs without leaves. Through a different analysis, Knauer et al. [9] showed that Salamon's algorithm in [15] can actually take  $O(n^3)$  time to achieve a performance ratio  $\frac{5}{3}$  even on undirected simple graphs.

Salamon et al. [15] also studied the vertex-weighted cases of MIST which asks for a maximum weighted spanning tree of a vertex weighted graph. They gave a  $O(n^4)$ -time and  $(2\Delta - 3)$ -approximation algorithm for weighted MIST on graphs without leaves, where  $\Delta$  is the maximum degree of the graph. They also gave a  $O(n^4)$ -time and 2-approximation algorithm for weighted MIST on claw-free graphs without leaves. Later, Knauer et al. [9] presented a  $(3 + \epsilon)$ -approximation algorithm for weighted MIST on undirected simple graphs.

Fixed parameter algorithms of MIST have also been extensively studied in the recent years. Prieto and Sloper [11] designed the first FPT-algorithm with running time  $O^*(2^{4k \log k})$  in 2003. Coben et al. [3] improved this algorithm to achieve a time complexity  $O^*(49.4^k)$ . Then an FPT-algorithm for MIST with time complexity  $O^*(8^k)$  was proposed by Fomin et al. [6], who also gave an FPT-algorithm for its directed version with time complexity  $O^*(16^{k+o(k)})$  [5]. For directed graphs, a randomized FPT algorithm proposed by M. Zehavi is by now the fastest one which runs in  $O^*(2^{(2 - \frac{\Delta+1}{\Delta(\Delta-1)})k})$  time [18], where  $\Delta$  is the vertex degree bound of a graph. On cubic graphs in which each vertex has degree three, Binkele-Raible et al. [2] proposed an  $O^*(2.1364^k)$  time algorithm.

For the kernalization of MIST, Prieto and Sloper first presented an  $O(k^3)$ -vertex kernel [11, 12]. Later, they improved it to  $O(k^2)$  [13]. Recently, Fomin et al. [6] gave a  $3k$ -vertex kernel for this problem, which is the best by now.

As for the exact exponential algorithms to solve MIST, Binkele-Raible et al. [2] proposed a dynamic programming algorithm with time complexity  $O^*(2^n)$ . Moreover, on graphs with maximum vertex degree bounded by 3, they devised a branching algorithm with  $O(1.8612^n)$  time and polynomial space.

Several years have passed since the  $\frac{5}{3}$ -approximation algorithm for MIST was proposed. In this paper, we devote to approximate MIST to a better performance ratio. All our progresses are based on a new observation for bounding the number of the internal vertices in a spanning tree, which shows that any spanning tree has less internal vertices than the edges a maximum path-cycle cover has. Thus starting with a maximum path-cycle cover, we can devise a 1.5-approximation algorithm for MIST on undirected simple graphs. By a slight modification of the 1.5-approximation algorithm, we can devise a  $\frac{4}{3}$ -approximation algorithm for MIST on graphs without leaves.

This paper is organized as follows. Section 2 presents the concepts and notations related to path cover, path-cycle cover and maximum internal spanning tree on graphs. Section 3 uses a maximum path-cycle cover of a graph to bound the number of internal vertices in a spanning tree of that graph. Section 4 shows that a graph can be pruned by removing some of its leaves so that its maximum internal spanning tree can have as many internal vertices as those the maximum internal spanning tree of the original graph has. In section 5, we present an algorithm for MIST to achieve the performance ratio  $\frac{3}{2}$ . In section 6, we present an algorithm for MIST on graphs without any leaves to achieve the performance ratio  $\frac{4}{3}$ . Section 7 is concluded by looking forward to the future work for MIST.

## 2 Preliminaries

In this paper, all graphs in which we are going to find spanning trees are undirected, simple and connected. Each path or cycle in a graph is always simple. The first and the last vertices of a path are the *endpoints* of that path, while the others except the endpoints of a path are the *inner* vertices of that path. The *length* of a path or cycle is the number of edges in it. A *connected component* of a graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph. A connected component of a graph is referred to as a *path (cycle) component* if it is a path (cycle) of that graph. A vertex in a graph is a *leaf* if its degree is 1, and *internal* if its degree is larger than 1. Two vertices, say  $u$  and  $v$ , are adjacent *respecting an edge of a graph* if  $(u, v)$  is an edge of that graph.

Let  $G = (V, E)$  be an undirected simple graph. For  $V_1 \subseteq V$ , a subgraph of  $G$  is *induced by*  $V_1$  if it has the vertex set  $V_1$  and all the edges of  $G$  with both ends in  $V_1$ . The subgraph of  $G$  induced by  $V_1$  is denoted by  $G[V_1]$ . A subgraph of  $G$  is referred to as a *spanning subgraph* of  $G$  if it has the vertex set  $V$  and the edge set  $E_1 \subseteq E$ .

A spanning subgraph of  $G$  is a *path-cycle cover* of  $G$  if every vertex in it is incident with at most 2 edges. A path-cycle cover of  $G$  is *maximum* if its number of edges are maximized over all path-cycle covers of  $G$ . A spanning subgraph of  $G$  is a *path cover* if every connected component of it is a path. A path cover of  $G$  is *maximum* if its number of edges are maximized over all path covers of  $G$ .

A maximum path cover of a graph is also a path-cycle cover. Since a path-cycle cover can be found for every graph in polynomial time [4, 17], the number of edges in a path cover of a graph can be bounded by the following lemma.

**Lemma 1.** *There are not more edges in a maximum path cover of a graph than those in a maximum path-cycle cover of that graph.*

A *maximum internal spanning tree* of  $G$  is a spanning tree of  $G$  whose number of internal vertices are maximized over all spanning trees of  $G$ . The *Maximum Internal Spanning Tree* problem, MIST namely, is given by an undirected simple graph, and asks to find a maximum internal spanning tree for that graph.

## 3 Bounding the Number of Internal Vertices in a Spanning Tree

Let  $G$  be an undirected simple graph instead of a tree. In this section we show that the number of internal vertices in a spanning tree of  $G$  can be bounded by the number of edges in a maximum path-cycle cover of  $G$ .

**Lemma 2.** *If a tree has more than one vertex, then there is a path cover of the tree such that the path cover has less path components than the leaves that tree has. (The proof is omitted.)*

By Lemma 2, the number of leaves in a spanning tree can be bounded by,

**Lemma 3.** *A maximum internal spanning tree of  $G$  has more leaves than the path components a maximum path cover of  $G$  has. (The proof is omitted.)*

This lemma leads to an upper bound for the number of internal vertices a spanning tree of  $G$  has. That is,

**Theorem 1.** *A maximum internal spanning tree of  $G$  has less internal vertices than the edges a maximum path cover of  $G$  has. (The proof is omitted.)*

Recall by Lemma 1 that a maximum path cover of  $G$  cannot have more edges than those a maximum path-cycle cover of  $G$  has. Thus,

**Corollary 1.** *A maximum internal spanning tree of  $G$  has less internal vertices than the edges a maximum path-cycle cover of  $G$  has.*

Inspired by Corollary 1, we can start with a maximum path-cycle cover of a graph to construct a spanning tree of that graph for the hope that a good performance ratio can be achieved. . In order to make it possible to construct a spanning tree with at least two thirds times as many internal vertices as the edges a maximum path-cycle cover has, we have to prune some leaves away from  $G$ . The pruning for a graph will be stated in the next section.

## 4 Pruning a Graph

In this section,  $G$  is supposed to be an undirected simple graph instead of a tree. We show that  $G$  can be pruned by removing some of its leaves, thus into such a special subgraph of  $G$  that a maximum internal spanning tree of that subgraph has not less internal vertices than those a maximum internal spanning tree of  $G$  has. Therefore, we just need to pay attention to those special graphs for finding spanning trees of them. The pruning for  $G$  is derived from the following lemma.

**Lemma 4.** *If in  $G$ , two leaves, say  $v_1, v_2$ , both are adjacent to the same internal vertex respecting an edge of  $G$ , then the subgraph of  $G$  induced by  $V(G) - \{v_1\}$  or  $V(G) - \{v_2\}$  must have a spanning tree which has the same set of internal vertices as that a maximum internal spanning tree of  $G$  has. (The proof is omitted.)*

If two or more leaves are adjacent to one internal vertex of  $G$ , then by Lemma 4, all but one of them can be removed safely for finding a maximum internal spanning tree of it. This is the so called *pruning* for  $G$ . That is,

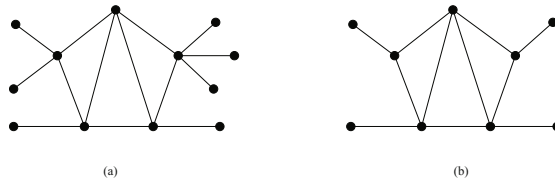
**Corollary 2.** *There exists such a subgraph of  $G$  that (1) each internal vertex is adjacent to at most one leaf respecting an edge of the subgraph; (2) a maximum internal spanning tree of it has not less internal vertices than those a maximum internal spanning tree of  $G$  has. (The proof is omitted.)*

In what follows, the *pruning* for  $G$  refers to removing some of its leaves so that the subgraph satisfies Corollary 2. If  $G_1$  is the subgraph resulted from the pruning for  $G$ , then each spanning tree of  $G_1$  can be transformed into a spanning tree of  $G$  by adding those leaves removed by the pruning for  $G$ . That is,



**Lemma 5.** *For any spanning tree of  $G_1$ , there is a spanning tree of  $G$  which has not less internal vertices than those the spanning tree of  $G_1$  has. (The proof is omitted.)*

A graph is *well-pruned* if every internal vertex of it is adjacent to at most one leaf respecting an edge of the graph. As an example, Fig. 1 presents a graph as well as a well-pruned subgraph of it. Let  $G_1$  be the graph resulted from the pruning for  $G$ . Then  $G_1$  must be well-pruned. Let  $T_1$  be a spanning tree of  $G_1$ . A spanning tree, say  $T$ , of  $G$  can be constructed by adding those leaves removed in the pruning process. Recall that  $T$  satisfies Lemma 5. Let respectively,  $I(T)$ ,  $I(T_1)$  be the sets of internal vertices in  $T$  and  $T_1$ , while  $I(G)$ ,  $I(G_1)$  be the sets of internal vertices in the maximum internal spanning trees of  $G$  and  $G_1$ . Then  $\frac{|I(G)|}{|I(T)|} \leq \frac{|I(G_1)|}{|I(T_1)|}$  follows from Corollary 2 and Lemma 5. In other words, if one can design an algorithm for MIST on the well-pruned graphs with some substantial performance ratio, one can design an algorithm for MIST on all graphs with the same performance ratio. Thus in the next section, we only focus on the well-pruned graphs to ask for their spanning trees.



**Fig. 1.** The pruning for  $G$ . (a) A graph ( $G$ ) in which one internal vertex can be adjacent to more than one leaf. (b) A well-pruned subgraph ( $G_1$ ) resulted from the pruning for  $G$ .

## 5 Finding a Spanning Tree of a Well-Pruned Graph

Our algorithm for MIST starts with an arbitrary maximum path-cycle cover of a well-pruned graph. In this section,  $G_1$  always stand for a well-pruned graph. We aim to assemble the connected components in a maximum path-cycle cover into a spanning tree of  $G_1$  which has at least two thirds as many internal vertices as those a maximum internal spanning tree of  $G_1$  has. To meet this aim, we have to construct a spanning forest of  $G_1$ , such that each subtree of the forest is made from a group of connected components in the maximum path-cycle cover of  $G_1$ , and has at least two thirds as many internal vertices as the edges these connected components have. For convenience to construct such a forest, we try to reconstruct a maximum path-cycle cover at first.

### 5.1 Reconstructing a Maximum Path-Cycle Cover

We also treat a maximum path-cycle cover as a set of cycle components and path components. A path component is a *singleton* if its length is zero. Following the notions in Section 2, a vertex of a path component is inner if its degree in it is 2, and an endpoint otherwise. In particular the *endpoint* of a singleton is the singleton itself. Note that although two vertices in distinct connected components in a maximum path-cycle cover are not adjacent respecting any edge in the maximum path-cycle cover, they can be *adjacent* respecting an edge of  $G_1$ .

In this section, a maximum path-cycle cover is always supposed to have more than one connected component. Otherwise, a genuine maximum internal spanning tree can be trivially obtained from that maximum path-cycle cover. Since  $G_1$  is connected, every connected component of a maximum path-cycle cover of  $G_1$  must have at least one vertex which is adjacent to a vertex outside of it respecting an edge of  $G_1$ .

Moreover, one can even find that a maximum path-cycle cover can be transformed into one in which every path component of length no larger than 2 must have one endpoint adjacent to a vertex outside of it respecting an edge of  $G_1$ . That is,

**Lemma 6.** *There must be a maximum path-cycle cover of  $G_1$  in which, if a path component is of length no larger than 2, then respecting an edge of  $G_1$ , at least one endpoint of it is adjacent to a vertex which falls outside of it. (The proof is omitted.)*

Since an endpoint of a path component cannot be adjacent to an endpoint of another path component respecting an edge of  $G_1$ , we have a stronger statement than that of Lemma 6. That is,

**Lemma 7.** *There must be a maximum path-cycle cover of  $G_1$  in which, if a path component is of length no larger than 2, then respecting an edge of  $G_1$ , at least one endpoint of it is adjacent to a vertex of a cycle component or an inner vertex of another path component. (The proof is omitted.)*

If a path component has an endpoint adjacent to a vertex of a cycle component respecting an edge of  $G_1$ , they can be merged into a path component.

**Lemma 8.** *There is such a maximum path-cycle cover that no path component has an endpoint adjacent to a vertex of a cycle component respecting an edge of  $G_1$ . (The proof is omitted.)*

Due to Lemma 7, a path component of length 1 or 2 can be eliminated if it has an endpoint adjacent to an inner vertex of another path component of length 1 or 2 respecting an edge of  $G_1$ . That is,

**Lemma 9.** *There is a maximum path-cycle cover of  $G_1$  in which, if a path component is of length 1 or 2, then respecting an edge of  $G_1$ , at least one endpoint of it must be adjacent to an inner vertex of a path component of length larger than 2. (The proof is omitted.)*

By Lemma 6, 8 and 9, there is a reconstruction for any maximum path-cycle cover of  $G_1$  so that the resultant maximum path-cycle cover satisfies the following lemma.

**Lemma 10.** *There is a maximum path-cycle cover such that, (1) every singleton must be adjacent to an inner vertex of a path component respecting an edge of  $G_1$ ; (2) every path component of length 1 or 2 must have one endpoint adjacent to an inner vertex of a path component of length larger than 2 respecting an edge of  $G_1$ . (The proof is omitted.)*

In the next subsection, we'll build up a spanning tree of  $G_1$  from a maximum path-cycle cover for which Lemma 10 holds true.

### 5.2 Assembling a Maximum Path-Cycle Cover into a Spanning Tree

Let  $H_{new}$  be a maximum path-cycle cover of  $G_1$  for which Lemma 10 holds true. Let  $p, c$  be a path and a cycle component respectively in  $H_{new}$ , while  $T$  be a subtree of  $G_1$ . Then  $p$  joins  $T$ , if  $V(p) \subseteq V(T)$  and  $E(p) \subseteq E(T)$ , and  $c$  joins  $T$ , if  $V(c) \subseteq V(T)$  and  $|E(c) \cap E(T)| \geq |E(c)| - 1$ . We specially pay attention to those subtrees of  $G_1$  which are joined by at least one connected component in  $H_{new}$ . A connected component in  $H_{new}$  joins a sub-forest of  $G_1$ , if it joins some tree in this sub-forest. A subtree of  $G_1$  is  $\alpha$ -approximate ( $0 \leq \alpha \leq 1$ ), if it has at least  $\alpha$  times as many internal vertices as the edges of the connected components which join it. A sub-forest of  $G_1$  is  $\alpha$ -approximate ( $0 \leq \alpha \leq 1$ ), if all the trees in it are  $\alpha$ -approximate. In this subsection, we aim to assemble the connected components in  $H_{new}$  into a  $\frac{2}{3}$ -approximate spanning tree of  $G_1$ . To meet this aim, we show that the connected components in  $H_{new}$  can be assembled into such a  $\frac{2}{3}$ -approximate spanning forest of  $G_1$  that every connected component in  $H_{new}$  joins just one tree of it.

A path component of length larger than 2 must be  $\frac{2}{3}$ -approximate. Thus,

**Lemma 11.** *There is a  $\frac{2}{3}$ -approximate sub-forest of  $G_1$ , such that every path component of length larger than 2 in  $H_{new}$  joins one tree of it. (The proof is omitted.)*

Those singletons and path components of length 1 or 2 in  $H_{new}$  can be assembled together with the path components of length larger than 2 respectively, thus into a  $\frac{2}{3}$ -approximate forest of larger size. That is,

**Lemma 12.** *There is a  $\frac{2}{3}$ -approximate sub-forest of  $G_1$ , such that every path component in  $H_{new}$  joins one tree of it. (The proof is omitted.)*

The remainder is to construct a  $\frac{2}{3}$ -approximate forest such that all connected components in  $H_{new}$  join it.

**Lemma 13.** *There is a  $\frac{2}{3}$ -approximate sub-forest of  $G_1$ , such that every connected component in  $H_{new}$  joins one tree of it. (The proof is omitted.)*

By Lemma 11, 12 and 13, a  $\frac{2}{3}$ -approximate forest can be constructed with each connected component in  $H_{new}$  joining just one tree of it. Namely, this forest must be a spanning forest of  $G_1$ . Since  $G_1$  is connected, we can use a set of edges of  $G_1$  to link the trees in the forest into a spanning tree of  $G_1$ , which has not less internal vertices than those all trees in the forest have. Finally, we summarize for finding a spanning tree of a well-pruned graph as in Algorithm 1.

---

**Algorithm 1.** Finding a maximum internal spanning tree of a well-pruned graph

---

**Input:**

$G_1$ : a graph with each internal vertex adjacent to at most one leaf.

**Output:**

A spanning tree of  $G_1$ .

- 1: Find a maximum path-cycle cover  $H$  of  $G_1$ ;
  - 2: Reconstruct  $H$  into  $H_{new}$ ; (Lemma 6, 8, 9)
  - 3:  $F \leftarrow \{p \in H_{new} : p \text{ is a path component and } |E(p)| > 2\}$ ; (Lemma 11)
  - 4: Assemble all path components of length 0, 1 and 2 into  $F$ ; (Lemma 12)
  - 5: Assemble all cycle components into  $F$ ; (Lemma 13)
  - 6: Link the trees in  $F$  into a spanning tree of  $G_1$  and output it.
- 

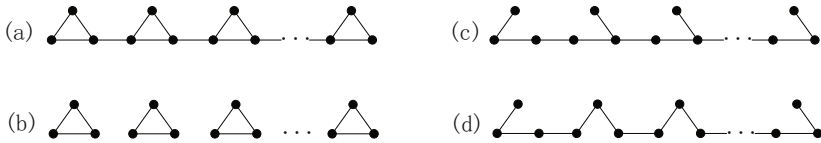
**Lemma 14.** *Algorithm 1 can always output a spanning tree of  $G_1$  which has at least two thirds as many internal vertices as those a maximum internal spanning tree of  $G_1$  has. (The proof is omitted.)*

Let  $V_1$  and  $E_1$  be the vertex set and the edge set of  $G_1$  respectively. It takes  $O(|V_1| |E_1|^{1.5} \log|V_1|)$  time to find a maximum path-cycle cover of  $G_1$  [17]; it takes  $O(|V_1| + |E_1|)$  time to reconstruct a maximum path-cycle cover. Thus, Step 1, 2 take  $O(|V_1| |E_1|^{1.5} \log|V_1|)$  time. Each step of 4 and 5 for assembling those connected components into a sub-forest of  $G_1$  takes  $O(|V_1| + |E_1|)$  time. To sum up, the time complexity of Algorithm 1 is  $O(|V_1| |E_1|^{1.5} \log|V_1|)$ .

Recall from section 4 that there is an  $r$ -approximation algorithm for MIST on general undirected simple graphs if an  $r$ -approximation algorithm exists for MIST on well-pruned graphs. Thus,

**Theorem 2.** *For any undirected simple graph, finding a maximum internal spanning tree can be approximated to a performance ratio 1.5 in polynomial time. (The proof is omitted.)*

*Tightness* Let's give an example in Fig. 2. For  $G_1$  given in Fig. 2(a) as an instance of MIST, if Algorithm 1 starts with a maximum path-cycle cover exactly containing  $k$  cycles of length 3, then it will return the spanning tree  $T$  with  $2k$  internal vertices while  $T^*$  is a maximum internal spanning tree with  $3k - 2$  internal vertices. Increasing  $k$ , we come close to a  $\frac{3}{2}$  ratio.



**Fig. 2.** (a) A well-pruned graph ( $G_1$ ). (b) A maximum path-cycle cover of  $G_1$  which has  $k$  triangles. (c) The spanning tree of  $2k$  internal vertices Algorithm 1 returns if starting with the maximum path-cycle cover in (b). (d) A maximum internal spanning tree of  $3k - 2$  internal vertices.

## 6 A $\frac{4}{3}$ -Approximation Algorithm for the Graphs without Leaves

MIST on graphs without any leaves is NP-hard because the Hamilton-path problem on 2-connected cubic bipartite planar graphs is NP-Hard [1], and a 2-connected cubic bipartite planar graph must be simple and have all its vertices each incident to at least 2 edges. A maximum path-cycle cover of a graph with  $n$  vertices and  $m$  edges can be found in  $O(n^2m)$  time [8], even if the maximum path-cycle cover is restricted to have each cycle component with at least 4 edges. In this section, a graph does not contain any leaves whenever it is mentioned, and each cycle component in a maximum path-cycle cover has at least 4 edges. We present an approximation algorithm for MIST and show that the algorithm can achieve the performance ratio  $\frac{4}{3}$  for a graph without any leaves. We denote by  $G'$  an undirected simple graph without any leaves in this section.

### 6.1 Reconstructing a Maximum Path-Cycle Cover

The reconstruction tries to transform an arbitrary maximum path-cycle cover into one in which every path component of length 1, 2, or 3 has at least one endpoint adjacent to an inner vertex of a path component of length at least 4 respecting an edge of  $G'$ .

Moreover, a maximum path-cycle cover of  $G'$  is supposed to have more than one connected component again, for the same reason as stated in Section 5.1. Since  $G'$  is connected, every connected component of a maximum path-cycle cover of  $G'$  must have at least one vertex which is adjacent to a vertex outside of it respecting an edge of  $G'$ .

Note that there must be a maximum path-cycle cover of  $G'$  for which Lemma 6 and 7 still hold, even if each cycle component in the maximum path-cycle cover has at least 4 edges. In order to reconstruct a maximum path-cycle cover of  $G'$ , the following two lemmas are necessary as supplements to Lemma 6 and 7.

**Lemma 15.** *There must be a maximum path-cycle cover of  $G'$  in which, if a path component is of length three, then respecting an edge of  $G'$ , at least one endpoint of it is adjacent to a vertex which falls outside of it. (The proof is omitted.)*

The lemma 15 can be strengthened by the following lemma.

**Lemma 16.** *There must be a maximum path-cycle cover of  $G'$  in which, if a path component is of length three, then respecting an edge of  $G'$ , at least one endpoint of it is adjacent to a vertex of a cycle component or an inner vertex of another path component. (The proof is omitted.)*

By Lemma 15 and 16, a maximum path-cycle cover can be reconstructed so that it subjects to,

**Lemma 17.** *There is such a maximum path-cycle cover of  $G'$  that, (1) every singleton must be adjacent to an inner vertex of a path component respecting an edge of  $G'$ ; (2) every path component of length 1, 2 or 3 has at least one endpoint adjacent to an inner vertex of a path component of length at least 4 respecting an edge of  $G'$ . (The proof is omitted.)*

In the next subsection, we'll use a maximum path-cycle cover for which Lemma 17 holds true to build up a spanning tree of  $G'$ .

### 6.2 Assembling a Maximum Path-Cycle Cover into a Spanning Tree

Let  $H'_{new}$  be a maximum path-cycle cover for which Lemma 17 holds true. We aim at assembling all the connected components in  $H'_{new}$  into a  $\frac{3}{4}$ -approximate spanning tree of  $G'$ . There will be a  $\frac{3}{4}$ -approximate spanning tree of  $G'$ , if there is a  $\frac{3}{4}$ -approximate spanning forest of  $G'$ . Actually, a  $\frac{3}{4}$ -approximate forest can be constructed by almost the same method as that in section 5.2. That is,

**Lemma 18.** *There is a  $\frac{3}{4}$ -approximate spanning forest of  $G'$ , such that every connected component in  $H'_{new}$  joins one tree of it. (The proof is omitted.)*

---

**Algorithm 2.** Finding a spanning tree of a graph without leaves

---

**Input:**

$G'$ : a graph without leaves.

**Output:**

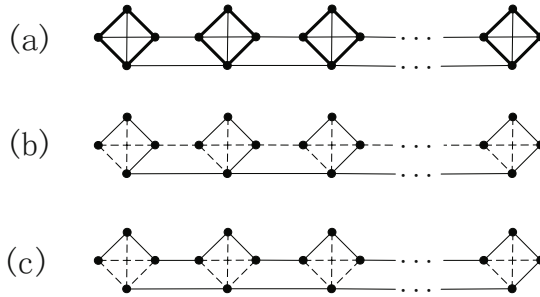
A spanning tree of  $G'$ .

- 1: Find a maximum path-cycle cover  $H'$  of  $G'$ ;
  - 2: Reconstruct  $H'$  into  $H'_{new}$ ; (Lemma 6, 8, 9, 17)
  - 3: Assemble the connected components in  $H'_{new}$  into a spanning forest  $F$ ; (Lemma 18)
  - 4: Link the trees in  $F$  into a spanning tree of  $G'$  and output it.
- 

Lemma 18 immediately implies a  $\frac{4}{3}$ -approximation algorithm for the MIST on graphs without leaves which is summarized in Algorithm 2

**Theorem 3.** *MIST can be approximated to within  $\frac{4}{3}$  in polynomial time for an undirected simple graph without leaves. (The proof is omitted.)*

*Tightness* In Fig. 3, we give an example to verify the performance of the algorithm. The  $\frac{4}{3}$ -approximation algorithm starts with a maximum path-cycle cover exactly containing  $k$  cycles of length 4 which is marked as bold edges in Figure 3(a). The algorithm will output  $T$  as its solution which is showed as solid edges in Figure 3(b), while  $T^*$  is a spanning tree as an optimal solution which is marked as solid edges in Figure 3(c). Since  $T$  has  $3k$  internal vertices, while  $T^*$  has  $4k - 2$  internal vertices, thus increasing  $k$ , we come close to a  $\frac{4}{3}$  ratio.



**Fig. 3.** (a)A graph ( $G'$ ) without leaves, where those  $k$  squares with bold edges serves as a maximum path-cycle cover. (b)The spanning tree of  $3k$  internal vertices the algorithm outputs, if it starts with a path-cycle cover of  $G'$  as in (a) of this figure. (c)A maximum internal spanning tree of  $4k - 2$  internal vertices.

## 7 Conclusions and Discussions

We have presented an algorithm for MIST which can achieve the performance ratio  $\frac{3}{2}$  on undirected simple graphs, and an algorithm for MIST which can achieve the performance ratio  $\frac{4}{3}$  on the graphs without leaves. We believe that the method of this paper can be used to design approximation algorithms for other problems whose solutions can be bounded by the size of the maximum path-cycle covers of graphs. It is interesting and open whether there is a polynomial algorithm with performance ratio equal to or less than  $\frac{4}{3}$  on undirected simple graphs.

**Acknowledgments.** This research is partially supported by national natural science foundation of China under the grant of 61472222 and natural science foundation of Shandong province of China under the grant of ZR2012FZ002.

## References

1. Akiyama, T., Nishizeki, T., Saito, N.: NP-completeness of the Hamiltonian cycle problem for bipartite graphs. *Journal of Information Processing* **3**(2), 73–76 (1980)
2. Binkele-Raible, D., Fernau, H., Gaspers, S., Liedloff, M.: Exact and parameterized algorithms for MAX INTERNAL SPANNING TREE. *Algorithmica* **65**, 95–128 (2013)

3. Coben, N., Fomin, F.V., Gutin, G., Kim, E.J., Saurabh, S., Yeo, A.: Algorithm for finding  $k$ -vertex out-trees and its application to  $k$ -internal out-branching problem. *JCSS* **76**(7), 650–662 (2010)
4. Edmonds, J., Johnson, E.L.: Matching: a well solved class of integer linear programs. In: *Combinatorial Structures and their Applications*, pp. 89–92. Gordon and Breach, New York (1970)
5. Fomin, F.V., Lokshtanov, D., Grandoni, F., Saurabh, S.: Sharp separation and applications to exact and parameterized algorithms. *Algorithmica* **63**(3), 692–706 (2012)
6. Fomin, F.V., Gaspers, S., Saurabh, S., Thomassé, S.: A linear vertex kernel for maximum internal spanning tree. *JCSS* **79**, 1–6 (2013)
7. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman (1979)
8. Hartvigsen, D.: Extensions of matching theory, Ph.D. Thesis. Carnegie-Mellon University (1984)
9. Knauer, M., Spoerhase, J.: Better Approximation Algorithms for the Maximum Internal Spanning Tree Problem. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) *WADS 2009*. LNCS, vol. 5664, pp. 459–470. Springer, Heidelberg (2009)
10. Lu, H., Ravi, R.: The power of local optimization approximation algorithms for maximum-leaf spanning tree. Technical report. Department of Computer Science, Brown University (1996)
11. Prieto, E., Sloper, C.: Either/Or: using VERTEX COVER structure in designing FPT-algorithms — the case of  $k$ -INTERNAL SPANNING TREE. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) *WADS 2003*. LNCS, vol. 2748, pp. 474–483. Springer, Heidelberg (2003)
12. Prieto, E.: Systematic kernelization in FPT algorithm design. Ph.D. Thesis, The University of Newcastle, Australia (2005)
13. Prieto, E., Sloper, C.: Reducing to independent set structure the case of  $k$ -internal spanning tree. *Nord. J. Comput.* **12**(3), 308–318 (2005)
14. Salamon, G., Wiener, G.: On finding spanning trees with few leaves. *Information Processing Letters* **105**(5), 164–169 (2008)
15. Salamon, G.: Approximating the Maximum Internal Spanning Tree problem. *Theoretical Computer Science* **410**(50), 5273–5284 (2009)
16. Salamon, G.: Degree-Based Spanning Tree Optimization. Ph.D. thesis, Budapest University of Technology and Economics, Hungary (2009)
17. Shiloach, Y.: Another look at the degree constrained subgraph problem. *Inf. Process. Lett.* **12**(2), 89–92 (1981)
18. Zehavi, M.: Algorithms for  $k$ -Internal Out-Branching. In: Gutin, G., Szeider, S. (eds.) *IPEC 2013*. LNCS, vol. 8246, pp. 361–373. Springer, Heidelberg (2013)



# Approximation Algorithms Inspired by Kernelization Methods

Faisal N. Abu-Khzam<sup>1</sup>(✉), Cristina Bazgan<sup>2,5</sup>,  
Morgan Chopin<sup>3</sup>, and Henning Fernau<sup>4</sup>

<sup>1</sup> Lebanese American University, Beirut, Lebanon  
`faisal.abukhzam@lau.edu.lb`

<sup>2</sup> PSL, University of Paris-Dauphine, LAMSADE UMR 7243, Paris, France  
`bazgan@lamsade.dauphine.fr`

<sup>3</sup> Institut für Optimierung und Operations Research, Universität Ulm, Ulm, Germany  
`morgan.chopin@uni-ulm.de`

<sup>4</sup> Fachbereich 4, Informatikwissenschaften, Universität Trier, Trier, Germany  
`fernau@uni-trier.de`

<sup>5</sup> Institut Universitaire de France, Paris, France

**Abstract.** Kernelization algorithms in the context of Parameterized Complexity are often based on a combination of reduction rules and combinatorial insights. We will expose in this paper a similar strategy for obtaining polynomial-time approximation algorithms. Our method features the use of approximation-preserving reductions, akin to the notion of parameterized reductions. We exemplify this method to obtain the currently best approximation algorithms for HARMLESS SET, DIFFERENTIAL and MULTIPLE NONBLOCKER, all of them can be considered in the context of securing networks or information propagation.

## 1 Introduction

In this paper, for the purpose of illustrating our method, we will mainly deal with maximization problems that are obtained from domination-type graph problems. We first describe these problems, using standard graph-theoretic terminology.

Let  $G = (V, E)$  be an undirected graph and  $D \subseteq V$ .

1.  $D$  is called a *dominating set* if, for all  $x \in V \setminus D$ , there is a  $y \in D \cap N(x)$ .  $V \setminus D$  is known as an *enclaveless set* [28] or as a *nonblocker set* [17].
2.  $D$  is called a *total dominating set* if, for all  $x \in V$ , there is a  $y \in D \cap N(x)$ .  $V \setminus D$  has been introduced as a *harmless set* or *robust set* (with unanimity thresholds) in [6].
3. If  $D$  can be partitioned as  $D = D_1 \cup D_2$  such that, for all  $x \in V \setminus D$ , there is a  $y \in D_2 \cap N(x)$ , then  $(D_2, D_1)$  defines a *Roman domination function*  $f_{D_1, D_2} : V \rightarrow \{0, 1, 2\}$  such that  $f_{D_1, D_2}(V) = 2|D_2| + |D_1|$ . According to [10],  $|V| - f_{D_1, D_2}(V)$  is also known as the *differential* of a graph (as introduced in [25]) if  $f_{D_1, D_2}(V)$  is smallest possible.
4. If for all  $x \in V \setminus D$ , there are  $k$  elements in  $D \cap N(x)$ , then  $D$  is a *k-dominating set*, see [14, 16, 21]. We will call  $V \setminus D$  a *k-nonblocker set*.

The maximization problems derived from these four definitions are: NON-BLOCKER, HARMLESS SET, DIFFERENTIAL, and  $k$ -NONBLOCKER. Although these problems are all better known from the minimization perspective, there is a good reason to study them in this complementary way: All of these minimization problems do not possess constant-factor approximations under reasonable complexity assumptions (the reduction shown in [15] for (TOTAL) DOMINATING SET starts from SET COVER), while the complementary problems can be treated in this favorable way. For ROMAN DOMINATION, observe that the reduction shown in [20] works from SET COVER, so that again (basically) the same lower bounds follow. This move is related to *differential approximation* [3]. Notice that this comes along with similar properties from the perspective of Parameterized Complexity: While natural parameterizations of the minimizations lead to W[2]-hard problems [18, 20], the natural parameterizations of the maximization counterparts are fixed-parameter tractable. However, as this is more customary as a combinatorial entity, let us refer (as usual) by  $\gamma(G)$  to the size of the smallest dominating set of  $G$ , by  $\gamma_t(G)$  to the size of the smallest total dominating set, by  $\gamma_R(G)$  to the Roman domination number of  $G$ , i.e., the smallest value of a Roman domination function of  $G$ , and by  $\gamma_k(G)$  to the size of the smallest  $k$ -dominating set of  $G$ .

*Some graph-theoretic notations.* Let  $G = (V, E)$  be a simple undirected graph. We denote by  $N(x)$  the set of neighbors of vertex  $x$ ; the cardinality of  $N(x)$  is the *degree* of  $x$ . A vertex of degree zero is known as an *isolated vertex*, and a vertex of degree one as a *leaf*. The number of vertices of a graph is called its *order*. Given  $U \subseteq V$ ,  $G[U]$  denotes the subgraph induced by  $U$ . A repetition-free sequence  $x_1, \dots, x_k$  of vertices is a *path* in  $G$  (of length  $k - 1$ ) if  $x_i x_{i+1} \in E$  for  $i = 1, \dots, k - 1$ . A *chain* is an induced path whose interior vertices are of degree two in  $G$ . The *diameter* of  $G$  is the greatest length of a shortest path in  $G$ .

*Main Results.* We introduce a notion of approximation-preserving reductions analogous to parameter-preserving reductions known in Parameterized Complexity in order to obtain new approximation algorithms. We introduce a general methodology to obtain constant-factor approximations for various problems. For instance, along with an algorithmic version of the upper bound obtained in [24] on the size of a total dominating set, we present a factor-two approximation algorithm for HARMLESS SET, beating the previously known factor of three [6]. Moreover, we are deriving a factor- $\frac{11}{3}$  approximation algorithm for DIFFERENTIAL, which was set up as an open problem in [9], where this approximability question could be only settled for bounded-degree graphs; our approach also improves on the factor-4 approximation exhibited in [7]. Finally, we present constant-factor approximation algorithms for  $k$ -NONBLOCKER.

*Organization of the paper.* Section 2 explains the use of reduction rules within maximization problems. It also exhibits the general method. Section 3 shows how to employ our general method to one specific problem in a non-trivial way.

Sections 4 and 5 show that the same method can be also applied to other problems. We conclude with discussing further directions of research.

All proofs and some more details that are omitted due to space restrictions can be found in the long version of this paper [1].

## 2 Approximation Preserving Reductions for Maximization Problems

A maximization problem  $\mathcal{P}$  can be specified by a triple  $(I_{\mathcal{P}}, \text{SOL}_{\mathcal{P}}, m_{\mathcal{P}})$ , where

1.  $I_{\mathcal{P}}$  is the set of input instances of  $\mathcal{P}$ ;
2.  $\text{SOL}_{\mathcal{P}}$  is a function that associates to  $x \in I_{\mathcal{P}}$  the set  $\text{SOL}_{\mathcal{P}}(x)$  of feasible solutions of  $x$ ;
3.  $m_{\mathcal{P}}$  provides on  $(x, y)$ , where  $x \in I_{\mathcal{P}}$  and  $y \in \text{SOL}_{\mathcal{P}}(x)$ , a positive integer which is the value of the solution  $y$ .

An optimum solution  $y^*$  to  $x$  satisfies: (i)  $y^* \in \text{SOL}_{\mathcal{P}}(x)$ , and (ii)  $m_{\mathcal{P}}(y^*) = \max\{m_{\mathcal{P}}(y) \mid y \in \text{SOL}_{\mathcal{P}}(x)\}$ . The value  $m_{\mathcal{P}}(y^*)$  is also referred to as  $m_{\mathcal{P}}^*(x)$  for brevity. The subscript  $\mathcal{P}$  will be dropped when no ambiguity exists.

Given a maximization problem  $\mathcal{P}$ , a *factor- $\alpha$  approximation*,  $\alpha \geq 1$ , associates to each  $x \in I_{\mathcal{P}}$  some  $y \in \text{SOL}_{\mathcal{P}}(x)$  such that  $\alpha \cdot m_{\mathcal{P}}(x, y) \geq m_{\mathcal{P}}^*(x)$ . A solution  $y \in \text{SOL}_{\mathcal{P}}(x)$  satisfying  $\alpha \cdot m_{\mathcal{P}}(x, y) \geq m_{\mathcal{P}}^*(x)$  is also called an  *$\alpha$ -approximate solution* for  $x$ .

We are now going to present a first key notion for this paper.

**Definition 1.** *An  $\alpha$ -preserving reduction, with  $\alpha \geq 1$ , is a pair of mappings  $\text{inst}_{\mathcal{P}} : I_{\mathcal{P}} \rightarrow I_{\mathcal{P}}$  and  $\text{sol}_{\mathcal{P}}$  which, given  $y' \in \text{SOL}_{\mathcal{P}}(\text{inst}_{\mathcal{P}}(x))$ , produces some  $y \in \text{SOL}_{\mathcal{P}}(x)$  such that there are constants  $a, b \geq 0$  satisfying  $a \leq \alpha \cdot b$  and the following inequalities:*

1.  $m_{\mathcal{P}}^*(\text{inst}_{\mathcal{P}}(x)) + a \geq m_{\mathcal{P}}^*(x)$ ,
2. for each  $y' \in \text{SOL}_{\mathcal{P}}(\text{inst}_{\mathcal{P}}(x))$ , the corresponding solution  $y = \text{sol}_{\mathcal{P}}(y')$  satisfies:  $m_{\mathcal{P}}(\text{inst}_{\mathcal{P}}(x), y') + b \leq m_{\mathcal{P}}(x, y)$ .

When referring to this definition, we mostly explicitly specify the constants  $a$  and  $b$  for ease of verification. An important trivial example is given by a pair of identity mappings that are  $\alpha$ -preserving for any  $\alpha \geq 1$ . Notice that a similar notion has been introduced in the context of minimization problems in [12, 13, 22].

**Theorem 1.** *Let  $\mathcal{P} = (I_{\mathcal{P}}, \text{SOL}_{\mathcal{P}}, m_{\mathcal{P}})$  be some maximization problem. If the pair  $(\text{inst}_{\mathcal{P}}, \text{sol}_{\mathcal{P}})$  describes an  $\alpha$ -preserving reduction and if, given some instance  $x$ ,  $y' \in \text{SOL}_{\mathcal{P}}(\text{inst}_{\mathcal{P}}(x))$  is an  $\alpha$ -approximate solution for  $\text{inst}_{\mathcal{P}}(x)$ , then  $y = \text{sol}_{\mathcal{P}}(y')$  is an  $\alpha$ -approximate solution for  $x$ .*

*Proof.* We have to prove that  $\alpha \cdot m_{\mathcal{P}}(x, y) \geq m_{\mathcal{P}}^*(x)$ . Now,

$$\frac{m_{\mathcal{P}}^*(x)}{m_{\mathcal{P}}(x, y)} \leq \frac{m_{\mathcal{P}}^*(\text{inst}_{\mathcal{P}}(x)) + a}{m_{\mathcal{P}}(\text{inst}_{\mathcal{P}}(x), y') + b} \leq \frac{\alpha m_{\mathcal{P}}(\text{inst}_{\mathcal{P}}(x), y') + \alpha b}{m_{\mathcal{P}}(\text{inst}_{\mathcal{P}}(x), y') + b} = \alpha$$

as required. □

This shows that an  $\alpha$ -preserving reduction leads to a special AP-reduction as defined in [4]. But there, these reductions were mainly used to prove hardness results, as it is also the case of [22] that we already mentioned. However, we use this notion to obtain approximation algorithms.

The notion of an  $\alpha$ -preserving reduction was coined following the successful example of kernelization reductions known from Parameterized Complexity [18]. One of the nice features of those is that they are usually compiled from simpler rules that are often based on some applicability conditions. In the following, we describe that this also works out for approximation. We need two further notions to make this precise.

We call an  $\alpha$ -preserving reduction  $(\text{inst}_{\mathcal{P}}, \text{sol}_{\mathcal{P}})$  *strict* if  $|\text{inst}_{\mathcal{P}}(x)| < |x|$  for all  $x \in I_{\mathcal{P}}$ , and it is called *polynomial-time computable* if the two mappings comprising the reduction can be computed in polynomial time.

**Lemma 1.** *If  $(\text{inst}_{\mathcal{P}}, \text{sol}_{\mathcal{P}})$  and  $(\text{inst}'_{\mathcal{P}}, \text{sol}'_{\mathcal{P}})$  are two  $\alpha$ -preserving reductions, then the composition  $(i, s) := (\text{inst}_{\mathcal{P}} \circ \text{inst}'_{\mathcal{P}}, \text{sol}'_{\mathcal{P}} \circ \text{sol}_{\mathcal{P}})$  is also an  $\alpha$ -preserving reduction. If both  $(\text{inst}_{\mathcal{P}}, \text{sol}_{\mathcal{P}})$  and  $(\text{inst}'_{\mathcal{P}}, \text{sol}'_{\mathcal{P}})$  are strict (poly-time computable, resp.), then the composition  $(i, s)$  is strict (poly-time computable, resp.).*

Reductions are often described in some conditional form:

**if condition then do action**

Our previous considerations apply also for this type of conditioned reductions, apart from the fact that an instance may not change, assuming that the reduction was not applicable, which means that the condition was not true for that instance. Further discussions can be found in the long version of the paper [1].

The general strategy that we follow can be sketched as follows:

1. Apply (strict, poly-time computable)  $\alpha$ -preserving reduction rules as long as possible.
2. Possibly modify the resulting graph so that it meets some requirements from known combinatorial results on the graph parameter of interest.
3. Compute some solution for the modified graph that satisfies the mentioned combinatorial bounds.
4. Construct from this solution a good approximate solution for the original instance.

In order to illustrate the use of this strategy, let us elaborate on NON-BLOCKER, matching a result from [27]; the current record is given in [2]. This goes along the lines of the kernelization result by Dehne *et al.* [17], but kernelization needs no constructive proof of the combinatorial backbone result; the non-constructive proof of [26] is hence sufficient.

1. Delete all isolates. (If the resulting graph is of minimum degree at least two, we are ready to directly apply the algorithm of Nguyen *et al.* [27].)
2. Merge all leaf neighbors into a single vertex.
3. Delete all leaves but one, which is  $x$ . This yields the graph  $G$  of order  $n_G$ .
4. Create a copy  $G'$  of the graph  $G$ ; call the vertices in the new graph by priming the names of vertices of  $G$ . Let  $H$  be the graph union of  $G$  and  $G'$  plus the edge  $xx'$ .  $H$  is of minimum degree at least two by construction.
5. Take the algorithm of Nguyen *et al.* [27] to obtain a dominating set  $D_H$  of  $H$  satisfying  $|D_H| \leq \frac{2}{5}n_H$ . Should the solution  $D_H$  contain  $x$  or  $x'$ , it is not hard to modify it to contain the leaf neighbors  $y$  or  $y'$ , instead.
6. Hence,  $D_G = V_G \cap D_H$  is a dominating set for  $G$  with  $|D_G| \leq \frac{2}{5}n_G$ . Trivially,  $N_G = V_G \setminus D_G$  is a nonblocker solution for  $G$  that is  $\frac{5}{3}$ -approximate.
7. As the merging and deletion reductions are  $\alpha$ -preserving for each  $\alpha \geq 1$ , we can safely undo them and hence obtain a  $\frac{5}{3}$ -approximate solution for the original graph instance.

### 3 Harmless Set

We are now turning towards HARMLESS SET as the most elaborate example of our methodology. First, we are going to present the combinatorial backbone of our result. Let  $S_2(G)$  be the set all vertices of degree two within  $G$ .

**Theorem 2.** (Lam and Wei [24]) *Let  $G$  be a graph of order  $n_G$  and of minimum degree at least two such that  $G[S_2(G)]$  decomposes into  $K_1$ - and  $K_2$ -components. Then,  $\gamma_t(G) \leq n_G/2$ .*

The proof of this theorem is non-constructive, as it uses tools from extremal combinatorics. In [1], we show how to obtain a polynomial-time algorithm that actually computes a total dominating set (TDS)  $D$  with  $|D| \leq n_G/2$  under the assumptions of Theorem 2. Our approximation algorithm for HARMLESS SET is based on obtaining a (small enough) TDS in a graph  $H$  obtained from the input  $G$  after a number of modifications (mainly vertex deletions). In the reduction from  $G$  to  $H$ , we distinguish between the number of deleted vertices  $d$  (to get from  $G$  to  $H$ ) and the number of vertices  $a$  that are added to convert the TDS  $D_H$  to  $D_G$ .

**Theorem 3.** *Let  $G$  be a graph of order  $n_G$  and let  $H$  be a graph of order  $n_H$  obtained from  $G$  by deleting  $d$  vertices and possibly adding some edges. Let  $D_G$  and  $D_H$  be TDS solutions of  $G$  and  $H$  such that  $a = |D_G| - |D_H| \leq d$ . If  $|D_H| \leq c \cdot n_H$  and  $d \leq \gamma_t(G)$ , then  $V(G) \setminus D_G$  is a harmless set of  $G$  whose size  $n_G - |D_G|$  is within a factor of  $(1 - c)^{-1}$  from optimum.*

*Proof.* As  $n_H = n_G - d$ ,  $|D_G| = |D_H| + a \leq c(n_G - d) + a = cn_G + (a - cd) \leq cn_G + d - cd = cn_G + (1 - c)d \leq cn_G + (1 - c)\gamma_t(G)$ . Hence,  $n_G - |D_G| \geq n_G - cn_G - (1 - c)\gamma_t(G) = (1 - c)(n_G - \gamma_t(G))$ . This immediately yields an approximation factor of  $(1 - c)^{-1}$ . □

In the following, we will present reduction rules that produce a graph  $G$  with the property (\*) that each vertex of degree bigger than one has at most one leaf neighbor. The surgery that produces a graph  $H$  from  $G$  as indicated in Theorem 3 includes removing all  $d$  leaves and adding edges to ensure that  $H$  has minimum degree of two and satisfies that each component of  $H[S_2(H)]$  has diameter at most one. Notice that all leaf neighbors in  $G$  belong to some optimum TDS of  $G$  without loss of generality. Due to (\*),  $\gamma_t(G) \geq d$  as required. Moreover, given some TDS solution  $D_H$  for  $H$ , we can produce a valid TDS solution  $D_G$  for  $G$  by adding all  $d$  leaf neighbors to  $D_H$ . Notice that Theorem 3 leads to a factor-2 approximation algorithm for HARMLESS SET based on a polynomial-time, constructive version of Theorem 2.

Now, we list  $\alpha$ -preserving reductions for HARMLESS SET. All missing correctness proofs can be found in the long version of this paper [1]. We start with two very simple rules.

**Isolate Reduction.** If there is some isolated vertex, produce the instance  $(\{x\}, \emptyset)$  that has trivially no solution.

**Leaf Reduction.** If there are two leaf vertices  $u, v$  with common neighbor  $w$ , then delete  $u$ . (It would go into the harmless set.)

**Observation 4.** *The Leaf Reduction is  $\alpha$ -preserving for any  $\alpha \geq 1$ .*

Hence from now on, no vertex can have two leaf neighbors.

We shall use the term *chain* to denote a path whose interior vertices are of degree two in  $G$ . A chain with one leaf endpoint is a *pendant chain*. A *floating chain* is a chain with two leaves. A *support vertex* is a non-pendant endpoint of a pendant chain. Support vertices may have more than one pendant chain. We shall reduce the length of pendant chains to at most two, based on the following reduction rules.

**Floating Chain Reduction.** Delete all floating chains.

Clearly, a maximum harmless set (and a minimum TDS) can be computed in linear time on such trivial connected components. Hence, we can verify the definition with suitably chosen values for  $a = b$ , which proves:

**Observation 5.** *The Floating Chain Reduction is  $\alpha$ -preserving for any  $\alpha \geq 1$ .*

**Long Chain Reduction.** Assume that  $G$  is a graph that contains a path  $x - u - v - w - y$ , where  $u, v, w$  are three consecutive vertices of degree two, where  $|N(y)| \geq 2$ . Then, construct the graph  $G'$  by merging  $x$  and  $y$  and deleting  $u, v, w$ .

**Theorem 6.** *The Long Chain Reduction is  $\alpha$ -preserving for any  $\alpha \geq 1$ .*

*Proof.* Let  $G$  be the original graph and  $G'$  the graph obtained from  $G$  by deleting the path  $u, v, w$  and merging  $x$  and  $y$  as described by the rule. We show that  $a = b = 2$  works out in our case by considering several cases.

(a) Let  $C$  be a maximum harmless set (HS) for  $G$ . The special case when  $N(x) = \{u\}$  is easy to verify. In the following discussion, we can hence assume that  $x$  has at least two neighbors. We now consider cases whether or not  $x \in C$  or  $y \in C$ .

(a1) Assume that  $x \in C$  and  $y \in C$ . Hence,  $u, w$  are not dominated neither by  $x$  nor by  $y$ . Since  $C$  is maximum, we can assume  $|C \cap \{u, v, w\}| = 1$ , as  $\min\{|N(x)|, |N(y)|\} \geq 2$ ; hence, if all of  $u, v$  and  $w$  are in  $V \setminus C$ , then we can replace  $w$  by another neighbor of  $y$  and obtain another optimum solution. Then,  $C' = C \setminus \{x, u, v, w\}$  is a HS of  $G'$ , with  $|C'| = |C| - 2$ .

(a2) Assume that  $x \notin C$  and  $y \notin C$ . First, let us discuss the possibility that  $u \notin C$  and  $w \notin C$ . As  $C$  is maximum, the purpose of this is to dominate (i)  $v$  and (ii)  $x$  and  $y$ . To accomplish (i), either  $u \notin C$  or  $w \notin C$  would suffice. However, as  $C$  is maximum, condition (ii) means that  $N(x) \setminus C = \{u\}$  and that  $N(y) \setminus C = \{w\}$ . By our assumptions,  $\min\{|N(x)|, |N(y)|\} \geq 2$ . Hence, there is a vertex  $z \in N(y)$ ,  $z \neq w$ . Now,  $\tilde{C} = (C \setminus \{z\}) \cup \{w\}$  is also a maximum HS satisfying  $\{v, w\} \subseteq \tilde{C}$ . From now on, we assume that  $|C \cap \{u, v, w\}| = 2$  and that  $|(N(x) \cup N(y)) \setminus (\{u, w\} \cup C)| \geq 1$ . Hence,  $C' = C \setminus \{u, v, w\}$  is a HS of  $G'$  with  $|C'| = |C| - 2$ .

(a3) Assume now that  $x \in C$  and  $y \notin C$ . (Clearly, the case that  $x \notin C$  and  $y \in C$  is symmetric.) As  $u$  is not dominated by  $x$ , either (i)  $\{u, v\} \subseteq V \setminus C$  or (ii)  $\{v, w\} \subseteq V \setminus C$ . In case (i),  $x$  is dominated by  $u$ , but  $y$  must (still) be dominated by some vertex from  $N(y) \setminus \{w\}$ . In case (ii), symmetrically  $y$  is dominated by  $w$ , but  $x$  must be dominated by some vertex from  $N(x) \setminus \{u\}$ . In both cases,  $\tilde{C} = (C \setminus \{x\}) \cup \{v\}$  is another maximum harmless set of  $G$ . This leads us back to the previous item (i.e.,  $|C'| = |C| - 2$ .)

Summarizing, we have shown that from  $C$  we can construct a harmless set  $C'$  for  $G'$  with  $|C'| = |C| - 2$ .

(b) Conversely, assume  $C'$  is some harmless set for  $G'$ . We distinguish two cases:

(b1) Assume that  $y \in C'$ . Then,  $y$  is dominated by some  $z$  in its neighborhood (in  $G'$ ). We consider two cases according to the situation in  $G$ . (i) If  $z \in N(x)$ , then  $C = C' \cup \{x, u\}$  is a HS in  $G$ . (ii) If  $z \in N(y)$ , then  $C = C' \cup \{x, w\}$  is a HS in  $G$ . In both cases,  $|C| = |C'| + 2$ .

(b2) If  $y \notin C'$ , then again  $y$  is dominated by some  $z$  in its neighborhood (in  $G'$ ). We perform the same case distinction as in the previous case: (i) If  $z \in N(x)$ , then  $C = C' \cup \{u, v\}$  is a HS in  $G$ . (ii) If  $z \in N(y)$ , then  $C = C' \cup \{v, w\}$  is a HS in  $G$ . In both cases,  $|C| = |C'| + 2$ .

(c) The reasoning from (b) shows that, if  $C$  is an optimum solution for  $G$ , then  $C'$  as obtained in part (a) of this proof is an optimum solution for  $G'$ .  $\square$

Similarly, one sees the correctness of the following rule.

**Cycle Chain Reduction.** If  $G$  is a graph that contains a cycle  $x - u - v - w - x$ , where  $u, v, w$  are three consecutive vertices of degree two, then construct  $G'$  by deleting  $u$ .

Finally, we deal with support vertices with multiple pendant chains. Assuming the Long Chain Reduction has been applied, any pendant chain is of length two or less. Accordingly, a support vertex where two or more pendant chains meet does belong to some optimum solution. The following rule makes this idea more precise.

**Pendant Chain Reduction.** Assume that  $G = (V, E)$  is a graph that contains two pendant chains with common endpoint  $v$  of which at least one path is of length two. Then, construct the graph  $G' = (V', E')$  by deleting one of the two pendant chains, keeping one which is of length two.

**Theorem 7.** *The Pendant Chain Reduction is  $\alpha$ -preserving for any  $\alpha \geq 1$ .*

We are now ready to apply Theorem 3. Assume the graph  $G = (V, E)$  is reduced according to the reduction rules described so far. Hence,  $G$  satisfies: (a)  $G$  contains no chain of three vertices of degree two. (b) By the Leaf Reduction rule, any vertex has at most one leaf neighbor. Let  $G'$  be a graph isomorphic to  $G$  so that each vertex  $v$  of  $G$  corresponds to a vertex  $v'$  of  $G'$ , under the assumed isomorphism  $f : V(G) \rightarrow V(G')$ . We construct a graph  $H$  obtained from the disjoint union of  $G$  and  $G'$  simply by adding edges between each leaf neighbor vertex  $v$  of  $G$  and  $v' = f(v) \in V(G')$ . Then, we remove all leaves.

Due to the application of Pendant Chain Reduction to  $G$  (and  $G'$ ), the addition of edges between corresponding leaf neighbors in  $G$  and  $G'$  does not introduce induced cycles with more than two consecutive degree-two vertices.

To the resulting graph  $H$ , apply Long Chain Reduction as long as possible. Notice that an application of this rule does never decrease degrees, adds two vertices to the solution and removes four vertices of the graph.

This results in a graph  $H'$  of order  $n_{H'}$  with minimum degree at least two containing no chain of three vertices of degree two. Hence, we can apply the (algorithmic) version of Theorem 2 that returns a TDS  $D_{H'}$  for  $H'$  with  $2|D_{H'}| \leq n_{H'}$ . Undoing the  $c$  Long Chain Reductions that we applied, we obtain a TDS  $D_H$  for  $H$  with  $2|D_H| = 2(|D_{H'}| + 2c) \leq n_{H'} + 4c = n_H$ . By symmetry, we can assume that  $|D_H \cap V(G)| \leq |D_H \cap V(G')|$ . Now, we add all support vertices to  $D_H \cap V(G)$  and further vertices to obtain  $D_G$  by the following rules:

- If a support vertex already belongs to  $D_H$ , then it could have been dominated via the edge that we introduced. As this interconnects to another support vertex, both already belonged to  $D_H$ . We arbitrarily select two neighbors (in  $G$ ) of these support vertices and put them into  $D_G$ . Hence, the mentioned support vertices and the attached leaves are totally dominated.
- If a support vertex  $x$  did not already belong to  $D_H$ , two cases arise: (a) If it was dominated (in  $H$ ) via an edge already belonging to  $G$ , then we do nothing on top of what we said. (b) If the support vertex  $x$  was dominated (in  $H$ ) by an edge  $xy$  not belonging to  $G$ , then we must add another neighbor  $z$  (in  $G$ ) of  $x$  to  $D_G$ . However, as (obviously) the vertex  $y$  belonged to  $D_H$  and was dominated by a neighbor (in  $G$ ) in  $D_H$ , we add (in total) two vertices  $x, z$  for the two support vertices  $x, y$ . Seen from the other side, this covers the



case of a support vertex that already belonged to  $D_H$  but was not dominated via the edge that we introduced.

Altogether, we see that we delete all leaves and introduce at most that many vertices into  $D_G$  (in comparison to  $D_H \cap V(G)$ ). By Theorem 3 and since all reduction rules take polynomial time, we obtain:

**Theorem 8.** HARMLESS SET is factor-2 polynomial-time approximable.

## 4 The Differential of a Graph

Let us start with an alternative presentation of this notion. Let  $G = (V, E)$  be a graph. For  $D_0 \subseteq V$ , let  $\partial(D_0) := |(\bigcup_{x \in D_0} N(x)) \setminus D_0| - |D_0|$ .  $\partial(D_0)$  is called the *differential* of the set  $D_0$ , and our aim is to find a vertex set that maximizes this quantity. This maximum quantity is known as the differential of  $G$ , written  $\partial(G)$ . The following combinatorial results are known:

**Theorem 9.** [8] Let  $G$  be a connected graph of order  $n$ . (a) If  $n \geq 3$ , then  $\partial(G) \geq n/5$ . (b) If  $G$  has minimum degree at least two, then  $\partial(G) \geq \frac{3n}{11}$ , apart from five exceptional graphs, none of them having more than seven vertices.

It is not hard to turn the first combinatorial result into a kernelization result, yielding a kernel bound of  $5k$ , where  $k$  is the natural parameterization of the DIFFERENTIAL. In [7], this result was improved to a kernel whose order is bounded by  $4k$ . This way, we can also get a factor-4 approximation. However, Theorem 9 suggests a possible improvement to a factor of  $\frac{11}{3}$  by our framework.

First, we have to show (see [1] more details) that the reduction rules presented in [7] as kernelization rules can be also interpreted as  $\alpha$ -preserving rules. We use some non-standard terminology. A *hair* is a sequence of two vertices  $uv$ , where  $u$  is a leaf and  $v$  has degree two. Then,  $u$  is also called a *hair leaf*.

**Lemma 2.** [7] Let  $G = (V, E)$  be a graph where none of the DIFFERENTIAL reduction rules listed in [1, 7] applies. Then,  $G$  has the following properties:

- (1) To each vertex, at most one leaf or one hair is attached, but not both together.
- (2) If we remove all leaves and all hairs from  $G$ , then the remaining graph  $\tilde{G} = (\tilde{V}, \tilde{E})$ , henceforth called nucleus, has minimum degree of at least two.
- (3) If a hair is attached to a vertex  $u$  in the nucleus, then no hair is attached to any neighbor of  $u$  within the nucleus.

We compute a sufficiently big solution for the nucleus and then use:

**Theorem 10.** Let  $G$  be a graph of order  $n_G$  and let  $H$  be a graph of order  $n_H$  obtained from  $G$  by deleting  $d$  vertices. Let  $D_G = D_{G,1} \cup D_{G,2}$  and  $D_H = D_{H,1} \cup D_{H,2}$  be Roman DS solutions of  $G$  and  $H$ , with  $D_{H,2} = D_{G,2}$  and  $a = |D_{G,1}| - |D_{H,1}| \leq d$ . If  $|D_{H,1}| + 2|D_{H,2}| \leq c \cdot n_H$  and  $d \leq \gamma_R(G)$ , then  $\partial(V(G) \setminus D_G) = n_G - 2|D_{G,2}| - |D_{G,1}|$  is within a factor of  $(1 - c)^{-1}$  from optimum.

We can turn the (non-constructive) combinatorial reasoning of [8] into a polynomial-time algorithm (see [1]), which allows us to conclude:

**Theorem 11.** DIFFERENTIAL is factor- $\frac{11}{3}$  polynomial-time approximable.

## 5 Multiple Nonblocker Sets

We shall assume  $k > 1$  in this section and, as usual, we consider a combinatorial upper bound on the size of some feasible solution of the minimization problem.

**Theorem 12** ([16]). *Let  $G$  be a graph of order  $n_G$  and a minimum degree at least  $k$ . Then  $\gamma_k(G) \leq \frac{k}{k+1}n_G$ .*

The known non-constructive proof can be turned into a polynomial-time algorithm (see [1]) computing a  $k$ -dominating set  $D$  with  $|D| \leq \frac{k}{k+1}n_G$ .

**Theorem 13.** *For a given graph  $G$  of order  $n_G$  and minimum degree at least  $k$ , one can compute a  $k$ -dominating set  $D$  with  $|D| \leq \frac{k}{k+1}n_G$  in polynomial time.*

Our approximation algorithm is based on obtaining a  $k$ -dominating set in a graph  $H$  obtained from the input  $G$  after adding the complete bipartite graph  $K_{k,k}$  and after a number of modifications.

**Theorem 14.** *Let  $G$  be a graph of order  $n_G$  and let  $H$  be a graph of order  $n_H$  obtained from  $G$  by deleting  $d$  vertices and adding  $2k$  new vertices,  $d > k$ . Let  $D_G$  and  $D_H$  be  $k$ -dominating set solutions of  $G$  and  $H$  such that  $a = |D_G| - |D_H| = d - k$ . If  $|D_H| \leq c \cdot n_H$  for some  $c < 1$  and  $d \leq \gamma_k(G)$ , then  $V(G) \setminus D_G$  is a  $k$ -nonblocker of  $G$  whose size  $n_G - |D_G|$  is within a factor of  $(1 - c)^{-1}$  from optimum.*

This result is understood modulo the additive constant  $k(2c - 1) < k$ .

Given a graph  $G$  as input, we construct a graph  $G'$  obtained from  $G$  by adding a complete bipartite graph  $K_{k,k}$  with (new) vertices  $u_1, \dots, u_k, v_1, \dots, v_k$ . This transformation is an L-reduction (as defined in [4]) and thus the approximation ratio is preserved. Now, we present reduction rules that when applied to  $G'$  produce a graph  $H$  with minimum degree at least  $k$ . Our reduction rules mainly deal with vertices of degree  $k - 1$  or less. We refer to such vertices as *low-degree* vertices. Each such vertex must be in any  $k$ -dominating set.

**Low-Degree Vertex Deletion.** If a low-degree vertex  $v$  has only low-degree neighbors, then delete  $v$ . If there is a vertex  $u$  with at least  $k + 1$  low-degree neighbors, then delete the edge between  $u$  and one low-degree neighbor of  $u$ .

**Observation 15.** *Low-Degree Vertex Deletion is  $\alpha$ -preserving for any  $\alpha \geq 1$ .*

**Low-Degree Merging.** Consider a graph that has been subject to the Low-Degree Vertex Deletion rule. For every vertex  $v$  of degree at least  $k$ , having  $q$  low-degree neighbors  $w_1, \dots, w_q$ , with  $q \leq k$ , connect  $v$  to  $v_1, \dots, v_q$  (from the  $K_{k,k}$  that was added as described above). Finally, delete all low-degree vertices.

**Observation 16.** *Low-Degree Merging is  $\alpha$ -preserving for any  $\alpha \geq 1$ .*

The reductions above take polynomial time, so that Theorem 14 allows us to conclude:

**Theorem 17.**  *$k$ -NONBLOCKER is factor- $(k + 1)$  polynomial-time approximable (modulo an additive constant less than  $k$ ).*

## 6 Conclusions

We presented a framework for obtaining approximations for maximization problems, inspired by similar reasonings for obtaining kernelization results. We see five directions from this approach:

- Paraphrasing [19], we might say that not only FPT, but also polynomial-time maximization is *P-time extremal structure*. This should inspire mathematicians working in combinatorics to work out useful bounds on different graph parameters. We started on domination-type parameters, and this might be a first venue of continuation, for example, along the lines sketched in [11, 23].
- Conversely, approximation algorithms that stay within the combinatorial grounds of their problem tend to reveal (combinatorial) insights into the problem that might get lost when moving for instance into the area of Mathematical Programming.
- The notion of  $\alpha$ -preserving reduction is similar to the local ratio techniques [5] that allowed to re-interpret many approximation algorithms (for minimization problems) in a purely combinatorial fashion. We see hope for similar developments using  $\alpha$ -preserving reduction for maximization problems.
- The fact that  $\alpha$ -preserving reductions are inspired by FPT techniques should allow to adapt these notions for parameterized approximation algorithms.
- Reductions are often close to practical heuristics and hence allow for fast implementations.

**Acknowledgments.** We are grateful for the support by the bilateral research cooperation CEDRE between France and Lebanon (grant number 30885TM).

## References

1. Abu-Khzam, F.N., Bazgan, C., Chopin, M., Fernau, H.: Data reductions and combinatorial bounds for improved approximation algorithms. Technical Report 1409.3742, ArXiv/CoRR (September 2014)
2. Athanassopoulos, S., Caragiannis, I., Kaklamanis, C., Kyropoulou, M.: An improved approximation bound for spanning star forest and color saving. In: Kráľovič, R., Niviński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 90–101. Springer, Heidelberg (2009)
3. Ausiello, G., Bazgan, C., Demange, M., Paschos, V.T.: Completeness in differential approximation classes. *Int. J. Found. Comp. Sci.* **16**(6), 1267–1295 (2005)
4. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation. In: *Combinatorial Optimization Problems and Their Approximability Properties*. Springer (1999)
5. Bar-Yehuda, R., Bendel, K., Freund, A., Rawitz, D.: Local ratio: a unified framework for approximation algorithms. *ACM Surv.* **36**(4), 422–463 (2004)
6. Bazgan, C., Chopin, M.: The robust set problem: Parameterized complexity and approximation. In: Rován, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 136–147. Springer, Heidelberg (2012)
7. Bermudo, S., Fernau, H.: Combinatorics for smaller kernels: The differential of a graph. To appear in *Theor. Comp. Sci.*

8. Bermudo, S., Fernau, H.: Lower bounds on the differential of a graph. *Disc. Math.* **312**, 3236–3250 (2012)
9. Bermudo, S., Fernau, H.: Computing the differential of a graph: hardness, approximability and exact algorithms. *Disc. Appl. Math.* **165**, 69–82 (2014)
10. Bermudo, S., Fernau, H., Sigarreta, J.M.: The differential and the Roman domination number of a graph. *Applic. Anal. & Disc. Math.* **8**, 155–171 (2014)
11. Borowiecki, M., Michalak, D.: Generalized independence and domination in graphs. *Disc. Math.* **191**, 51–56 (1998)
12. Brankovic, L., Fernau, H.: Parameterized approximation algorithms for HITTING SET. In: Solis-Oba, R., Persiano, G. (eds.) WAOA 2011. LNCS, vol. 7164, pp. 63–76. Springer, Heidelberg (2012)
13. Brankovic, L., Fernau, H.: A novel parameterised approximation algorithm for MINIMUM VERTEX COVER. *Theor. Comp. Sci.* **511**, 85–108 (2013)
14. Caro, Y., Roditty, Y.: A note on the  $k$ -domination number of a graph. *Int. J. Math. & Math. Sci.* **13**(1), 205–206 (1990)
15. Chlebík, M., Chlebíková, J.: Approximation hardness of dominating set problems in bounded degree graphs. *Inf. & Comput.* **206**, 1264–1275 (2008)
16. Cockayne, E.J., Gamble, B., Shepherd, B.: An upper bound for the  $k$ -domination number of a graph. *J. Graph Th.* **9**, 533–534 (1985)
17. Dehne, F., Fellows, M.R., Fernau, H., Prieto, E., Rosamond, F.A.: NONBLOCKER: parameterized algorithmics for MINIMUM DOMINATING SET. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 237–245. Springer, Heidelberg (2006)
18. Downey, R.G., Fellows, M.R.: *Fundamentals of Parameterized Complexity*. Springer (2013)
19. Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-time extremal structure I. In: *Algorithms and Complexity in Durham*, ACiD, pp. 1–41. King's College Publications (2005)
20. Fernau, H.: ROMAN DOMINATION: a parameterized perspective. *Int. J. Comp. Math.* **85**, 25–38 (2008)
21. Fink, J.F., Jacobson, M.S.:  $n$ -domination in graphs. In: *Graph Theory and Its Applications to Algorithms and Computer Science*, pp. 283–300. Wiley (1985)
22. Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar  $F$ -deletion: Approximation, kernelization and optimal FPT algorithms. In: 53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS, pp. 470–479. IEEE Computer Society (2012)
23. Kämmerling, K., Volkmann, L.: Roman  $k$ -domination in graphs. *J. Korean Math. Soc.* **46**, 1309–1318 (2009)
24. Lam, P.C.B., Wei, B.: On the total domination number of graphs. *Util. Math.* **72**, 223–240 (2007)
25. Mashburn, J.L., Haynes, T.W., Hedetniemi, S.M., Hedetniemi, S.T., Slater, P.J.: Differentials in graphs. *Util. Math.* **69**, 43–54 (2006)
26. McCuaig, B., Shepherd, B.: Domination in graphs of minimum degree two. *J. Graph Th.* **13**, 749–762 (1989)
27. Nguyen, C.T., Shen, J., Hou, M., Sheng, L., Miller, W., Zhang, L.: Approximating the spanning star forest problem and its application to genomic sequence alignment. *SIAM J. Comput.* **38**(3), 946–962 (2008)
28. Slater, P.J.: Enclaveless sets and MK-systems. *J. Res. Nat. Bur. Stand.* **82**(3), 197–202 (1977)

# An $5/4$ -Approximation Algorithm for Sorting Permutations by Short Block Moves

Haitao Jiang<sup>(✉)</sup>, Haodi Feng, and Daming Zhu

School of Computer Science and Technology, Shandong University,  
Jinan, People's Republic of China  
{htjiang, fenghaodi, dmzhu}@sdu.edu.cn

**Abstract.** Sorting permutations by short block moves is an interesting combinatorial problem derived from genome rearrangements. A short block move is an operation on a permutation that moves an element at most two positions away from its original position. The problem of sorting permutations by short block moves is to sort a permutation by the minimum number of short block moves. Our previous work showed that a special class of sub-permutations (named umbrella) can be optimally sorted in  $O(n^2)$  time. In this paper, we devise an  $5/4$ -approximation algorithm for sorting general permutations by short block moves, improving Heath's approximation algorithm with a factor  $4/3$  and our previous work with an approximation factor  $14/11$ . The key step of our algorithm is to decompose the permutation into a series of related umbrellas, then we can repeatedly exploit the polynomial algorithm for sorting umbrellas. To obtain the approximation factor of  $5/4$ , we also present an implicit lower bound of the optimal solution, which improves Heath and Vergara's result greatly.

## 1 Introduction

Sorting permutations by rearrangement operations has been studied widely during the last twenty years, since the three basic operations: reversals, transpositions and translocations was proposed by Sankoff [11].

A transposition is a rearrangement operation that operates on only one permutation. Precisely, a transposition cuts a segment out of the permutation and pastes it in a different location; i.e., it swaps two adjacent subpermutations. A transposition is also called a block-move. A lot of work was done for sorting a permutation by transpositions [1–4].

Actually, in the evolution process of the genomes, a segment is rarely moved far away from its original position. Naturally, Heath and Vergara proposed the problem of sorting by bounded block-moves [6], where the blocks must be moved within a bounded distance. A short block-move (also called 3-bounded transposition in [10]) is a transposition on a permutation such that the total length of the two segments swapped is at most three. Heath and Vergara presented an  $4/3$ -approximation algorithm for this problem, as well as polynomial algorithms for some special permutations [7]. Mahajan et al. simplified Heath and Vergara's

approximation algorithm, and described a linear-time algorithm to optimally sort the “correcting-hop-free” permutations [10]. Jiang et al. presented an  $O(n^2)$  algorithm to sort a special permutation with a structure called “umbrella”, then devised an  $(1 + \varepsilon)$ -approximation algorithm for permutations with many inversions [8], later, they also devised an 14/11-approximation algorithm for sorting general permutations by short block moves [9].

In this paper, we show an implicit lower bound of the short block move distance, and devise a new 1.25-algorithm approximating it within a factor 1.25.

## 2 Preliminaries

In the context of genome rearrangements, generally, genomes are represented by permutations, where each element stands for a gene. If there are  $n$  elements in a permutation, we denote them by the set  $I_n = \{1, 2, \dots, n\}$ . For example,  $\pi = [4, 1, 3, 2, 7, 5, 8, 6]$  is a permutation of eight elements. Let  $\iota_n = [1, 2, \dots, n-1, n]$  be the identity permutation of  $n$  elements. A block ( also called sub-permutation ) is a segment of contiguous elements or just one element. A short block is a block which contains at most two elements. A short block move operation swaps two adjacent short blocks, the total length of which is at most three. There are three possible forms of short block-moves:

1. *skip*:  $\rho([i], [i+1])$ , which exchanges the element  $\pi_i$  with the element  $\pi_{i+1}$ .
2. *right-hop*:  $\rho([i], [i+1, i+2])$ , which exchanges the element  $\pi_i$  with the block  $[\pi_{i+1}, \pi_{i+2}]$ , called a right hop of  $\pi_i$ .
3. *left-hop*:  $\rho([i, i+1], [i+2])$ , which exchanges the block  $[\pi_i, \pi_{i+1}]$  with the element  $\pi_{i+2}$ , called a left hop of  $\pi_{i+2}$  similarly.

For the sake of convenience, we also use  $\rho(\pi_i, \pi_{i+1})$  to denote a skip and  $\rho(\pi_i, \pi_{i+1} \pi_{i+2})$  (resp.  $\rho(\pi_i \pi_{i+1}, \pi_{i+2})$ ) to denote a right (resp. left) hop.

**Problem: Sorting by Short Block-Moves.**

**Input:** A permutation  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$ ,  $\pi_i \in I_n$ ,  $1 \leq i \leq n$ .

**Question:** Is there a sequence of short block-moves  $\rho_1, \rho_2, \dots, \rho_t$  such that  $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_t = \iota_n = [1, 2, \dots, n]$ , and  $t$  is minimized? The minimum integer  $t$  is the short block move distance of  $\pi$ .

An *inversion* in a permutation is a pair of elements  $\{\pi_i, \pi_j\}$  that are not in their correct relative order (i.e.,  $i < j$  and  $\pi_i > \pi_j$ ). There is no inversion in the identity permutation. A *correcting* short block-move corrects the relative order of the elements moved, i.e., a correcting skip erases a single inversion, while a correcting hop erases a pair of inversions.

Heath and Vergara showed that it suffices to consider only correcting short block-moves when seeking for an optimal sorting sequence of short block moves for a permutation. This result is summarized in the following Theorem.

**Theorem 1.** *For a permutation  $\pi$ , there exists an optimal sequence of short block-moves  $\rho_1, \rho_2, \dots, \rho_t$  that sorts  $\pi$  such that each short block move is a correcting short block move [6].*

In the following context of this paper, when referring to the optimal solution, we mean that all of its short block moves are correcting short block moves.

The following graph representation of a permutation serves as a fundamental tool for solving the sorting by short block moves problem. The permutation graph of  $\pi$  is a graph  $G(\pi) = (V, E)$ , where  $V = \{\pi_1, \pi_2, \dots, \pi_n\}$ ,  $E = \{(\pi_i, \pi_j) \mid i < j \text{ and } \pi_i > \pi_j\}$ . (Actually, the arcs in a permutation graph are directed, but as all the arcs direct from left to right, so we ignore its direction in the following context.)

Every arc of  $G(\pi)$  represents an inversion in  $\pi$ . Two arcs in the permutation graph are *compatible* if they share an identical endpoint. A *lone-arc* is an arc which does not share an identical endpoint with any arc.

As aforementioned, a correcting short block move corresponds to the removal of arcs in the permutation graph. a correcting Skip removes a single arc, and a correcting Hop removes two compatible arcs. If two arcs can be removed by a single Hop, we say that they form a *hop-match*. So, half of the number of arcs is a lower bound of the short block move distance.

### 3 An Implicit Lower Bound

In this section, we present a new lower bound for the number of short block moves to sort a permutation.

**Definition 1.** A permutation  $\pi = [\pi_1, \pi_2, \dots, \pi_m]$  is double increasing, if  $\pi$  can be completely decomposed into two disjoint sequences  $\pi_A = \pi_{p_1}, \pi_{p_2}, \dots, \pi_{p_i}$  and  $\pi_B = \pi_{q_1}, \pi_{q_2}, \dots, \pi_{q_j}$ , satisfying that

- $p_k < p_{k+1}, \pi_{p_k} < \pi_{p_{k+1}}, \text{ for } 1 \leq k \leq i - 1;$
- $q_r < q_{r+1}, \pi_{q_r} < \pi_{q_{r+1}}, \text{ for } 1 \leq r \leq j - 1;$
- $p_k \neq q_r, \text{ for } 1 \leq k \leq i \text{ and } 1 \leq r \leq j;$
- $\{\pi_{p_1}, \pi_{p_2}, \dots, \pi_{p_i}\} \cup \{\pi_{q_1}, \pi_{q_2}, \dots, \pi_{q_j}\} = \{\pi_1, \pi_2, \dots, \pi_m\}.$

A double increasing permutation has the following property.

*Property 1.* A permutation is double increasing if and only if in the corresponding permutation graph there is no vertex with non-zero in-degree and non-zero out-degree.

Actually, in the permutation graph of a double increasing permutation, either the in-degree of a vertex is zero, let it be in  $\pi_A$ ; or its out-degree is zero, let it be in  $\pi_B$ ; or both, let it be in either  $\pi_A$  or  $\pi_B$ . The lower bound of the number of short block moves for sorting a permutation is exactly followed by detecting double increasing sub-permutations in the input permutation, since sorting a double increasing sub-permutation must fulfill the following lemma.

**Lemma 1.** Given a permutation  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$ , if  $\pi_{i_1} < \pi_{i_2} < \dots < \pi_{i_l} < \pi_h$ , where  $h < i_1 < i_2 < \dots < i_l$ , then the arc  $(\pi_h, \pi_{i_j})$  can not form hop-match with  $(\pi_h, \pi_{i_k})$ , where  $|j - k| \geq 2$ , and  $1 \leq j, k \leq l$ .

*Proof.* W.L.O.G, assume to the contrary that  $(\pi_h, \pi_{i_j})$  and  $(\pi_h, \pi_{i_{j+r}})$  form a hop-match, where  $r \geq 2$  ( the case for  $r \leq -2$  is similar). Consider the scenario of the permutation  $\pi'$  immediately before that Hop can be applied,  $\pi' = [\dots, \pi_h, \pi_{i_j}, \pi_{i_{j+r}}, \dots]$ . Then, either  $\pi_{i_{j+1}}$  is located on the left of  $\pi_h$ , a new arc  $(\pi_{i_{j+1}}, \pi_{i_j})$  is generated; or  $\pi_{i_{j+1}}$  is located on the right of  $\pi_{i_{j+r}}$ , a new arc  $(\pi_{i_{j+r}}, \pi_{i_{j+1}})$  is generated. Both contradict to the assumption of Theorem 1 ( all short block moves are correcting in some optimal solution ).

**Corollary 1.** *Given a permutation  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$ , if  $\pi_h < \pi_{i_1} < \pi_{i_2} < \dots < \pi_{i_l}$ , where  $i_1 < i_2 < \dots < i_l < h$ , then the arc  $(\pi_{i_j}, \pi_h)$  can not form hop-match with  $(\pi_{i_k}, \pi_h)$ , where  $|j - k| \geq 2$ , and  $1 \leq j, k \leq l$ .*

Now, we construct the match graph  $G_\pi^M$ , which shows possible hop-matches, for a permutation  $\pi$ . Each edge of the permutation graph corresponds to a vertex in  $G_\pi^M$ . There is an edge between vertices if their corresponding edges in the permutation graph can form a hop-match.

Note that a maximum matching in the match graph is an upper bound of the number of hops while sorting the permutation by short block moves, and the unmatched vertices should be removed by Skips.

The match graph of a general permutation is actually the traditional *edge graph* of the permutation graph, but the match graph of a double increasing permutation is quite different, since it excludes some edges in light of Lemma 1 and Corollary 1. When drawing the match graph of a double increasing permutation, we list the vertices of  $\pi_B$  from left to right along the horizontal line in ascending order, and the vertices of  $\pi_A$  top-down along the vertical line in descending order. Then vertex corresponding to the edge  $(\pi_{p_i}, \pi_{q_j})$  is located on the intersection of the horizontal line of  $\pi_{p_i}$  and the vertical line of  $\pi_{q_j}$ , and its coordinates are  $(j, i)$ .

As the edges of a hop-match should share an endpoint in the permutation graph, the edges in the match graph are either horizontal or vertical. Let the length of the edge between  $(j, i)$  and  $(j, k)$  (resp.  $(k, i)$ ) be  $|i - k|$  (resp.  $|j - k|$ ). From Lemma 1 and Corollary 1, we have, each edge is of length one.

**Lemma 2.** *The match graph of a double increasing permutation is bipartite.*

Given the match graph  $G_\pi^M$  of a double increasing permutation  $\pi$ , after a bread-first search of  $G_\pi^M$ , we can denote  $G_\pi^M$  as  $(L_\pi^M, R_\pi^M, E_\pi^M)$ . Given a bipartite graph  $G = (L, R, E)$ , let  $X \subseteq L$  be a set of vertices, define  $NG(X) = \{v | (u, v) \in E, u \in X, v \in R\}$ .

**Lemma 3.** *A bipartite graph  $G = (L, R, E)$  has an  $L$ -saturating matching, if and only if  $|NG(X)| \geq |X|$ , for all  $X \subseteq L$ . [5]*

Since an unmatched vertex in the match graph corresponds to a Skip, the number of unmatched vertices is a lower bound on the number of Skips needed while sorting the permutation by short block moves. From Lemma 3, we have,



**Lemma 4.** *In the match graph  $G_\pi^M = (L_\pi^M, R_\pi^M, E_\pi^M)$  of a double increasing permutation  $\pi$ , for any  $X \subseteq L_\pi^M$ , if  $|NG(X)| < |X|$ , at least  $(|X| - |NG(X)|)$  skips are needed to remove the  $|X|$  arcs, while sorting  $\pi$  by short block moves.*

Given a general permutation, a maximal double increasing sup-permutation (abbreviated as *MDP*) is a double increasing sup-permutation, which satisfies: (1) the first element is of in-degree zero; (2) the last element is of out degree zero; (3) the other elements are either in-degree zero or out degree zero or both; (4) it is not contained in any other double increasing sup-permutation fulfilling (1),(2),(3). For example, in the permutation  $[4,2,1,6,8,3,5,11,7,10,9]$ ,  $[8,3,5,11,7]$  is double increasing but not a *MDP*, while  $[6,8,3,5,11,7]$  is a *MDP*. We can detect all maximal double increasing sub-permutations by scanning the permutation from left to right.

For these double increasing sub-permutations, some vertices in the match graph could possibly match to vertices not contained in the match graph, we will mark those vertices in the match graph.

**Lemma 5.** *In the match graph of a maximal double increasing sub-permutation of a permutation  $\pi$ , for any  $X \subseteq L_\pi^M$ , where each vertex  $x \in X$  is unmarked, if  $|NG(X)| < |X|$ , at least  $(|X| - |NG(X)|)$  skips are needed to eliminated the  $|X|$  arcs, while sorting  $\pi$  by short block moves.*

*Proof.* Since an unmarked vertex will either match to another vertex in the match graph or remain unmatched, then the lemma holds straightly due to Lemma 4.

The Skips detected by Lemma 5, can not be avoided in any optimal solution, we call them *necessary Skips*. It is obviously that the number of necessary Skips is a lower bound of the number of Skips the optimal solution.

## 4 An Equivalent Goal

Here, we present an equivalent condition to obtain the factor 5/4.

**Theorem 2.** *Let  $H^*$  and  $S^*$  be the number of Hops and Skips in the optimal solution respectively,  $H$  and  $S$  be the number of Hops and Skips of our algorithm respectively,  $S'$  be the number of necessary Skips. If  $(H + S')/(S - S') \geq 3/2$ , then  $(H + S)/(H^* + S^*) \leq 5/4$ .*

Therefore, to reach the ratio 5/4, it is sufficient for our algorithm to seek for enough Hops such that  $(H + S')/(S - S') \geq 3/2$ .

## 5 The Algorithm

### 5.1 Preprocessing

Firstly, we regulate the form of the input permutation  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$ . An *independent sub-permutation* is a sub-permutation  $\pi[i \rightarrow k] = [\pi_i, \pi_{i+1}, \dots, \pi_k]$

such that  $\pi_h < \pi_j < \pi_l$  for  $1 \leq h < i \leq j \leq k < l \leq n$ . A *minimal independent sub-permutation* is an independent sub-permutation, which does not contain any other independent sub-permutation. Since there is no arc between distinct minimal independent sub-permutations, we can sort each minimal independent sub-permutation independently. So it is sufficient to sort a minimal independent sub-permutation only.

In the following context of this paper, we assume the input permutation to be a minimal independent permutation.

Next, we show that some Hops must be in some optimal solution by checking the several left-most and right-most elements.

**Lemma 6.** *Let  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$  be a minimal independent permutation.*

- *If  $\pi_3 < \pi_2 < \pi_i$ , for all  $1 \leq i \leq n, i \neq 2, 3$ , then there exists some optimal solution containing the Hop  $\rho(\pi_1\pi_2, \pi_3)$ .*
- *If  $\pi_{n-2} > \pi_{n-1} > \pi_j$ , for all  $1 \leq j \leq n, j \neq n-2, n-1$ , then there exists some optimal solution containing the Hop  $\rho(\pi_{n-2}, \pi_{n-1}\pi_n)$ .*
- *If  $\pi_3 < \pi_1 < \pi_i$ , for all  $1 \leq i \leq n, i \neq 1, 3$ , then there exists some optimal solution containing the Hop  $\rho(\pi_1\pi_2, \pi_3)$ .*
- *If  $\pi_{n-2} > \pi_n > \pi_j$ , for all  $1 \leq j \leq n, j \neq n-2, n$ , then there exists some optimal solution containing the Hop  $\rho(\pi_{n-2}, \pi_{n-1}\pi_n)$ .*

In the following context of this paper, we assume that the input permutation  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$  is preprocessed, i.e.,  $\pi$  does not satisfy the conditions in Lemma 6.

### 5.2 Sort an Umbrella

The following structure in the permutation graph plays an important role for our algorithm.

**Definition 2.** *Let  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$ , the sub-permutation  $U = [\pi_h, \pi_{h+1}, \dots, \pi_{h+l}]$  (where  $1 \leq h \leq h+l \leq n$ ), satisfying that  $\pi_h > \pi_{h+j}$  for all  $1 \leq j \leq l$ , is called an umbrella.*

$\pi_h$  is called the *head* of the umbrella, and  $\pi_{h+j}$ 's ( $1 \leq j \leq l$ ) are called the *members* of the umbrella. The length  $L[U]$  of the umbrella  $U$  is the number of its members. We also call an umbrella of length  $l$  an *l-umbrella*.

In the permutation graph of an umbrella  $U$ , there are  $L[U]$  arcs connecting the head and the members, we call them *head-arcs*. There may also be arcs between members, we call them *member-arcs*. An umbrella containing only head-arcs are called *claw-umbrella*. Both head-arcs and member-arcs are called *inside-arcs*, denoted by  $E[U]$ . An umbrella  $U$  is *odd* (resp. *even*) if  $|E[U]|$  is odd (resp. even).

An umbrella  $U$  is *simple* if there is no correcting hop that can be applied to any three contiguous members in  $U$ . It is obvious that  $U$  is simple if  $L[U] \leq 3$ . From definition, we can convert a umbrella into a simple umbrella by applying hops between members greedily. Our previous work prove the following lemmas.

**Lemma 7.** *Given a simple umbrella  $U = [\pi_h, \pi_{h+1}, \dots, \pi_{h+l}]$ , either  $\pi_{h+1}$  or  $\pi_{h+2}$  is the minimum member. [8]*

**Lemma 8.** *Given any umbrella  $U$ , there exists  $\lfloor \frac{|E[U]|}{2} \rfloor$  hops and  $(|E[U]| - 2 \cdot \lfloor \frac{|E[U]|}{2} \rfloor)$  skips that sort  $U$ . [8]*

### 5.3 Sort Related Umbrellas

While sorting a single umbrella, we may need to apply a Skip provided that the umbrella is odd. To avoid this Skip, we attempt to sort a series umbrella together so that two Skips may be substituted by a Hop.

**Definition 3.** *The two simple umbrellas  $U_1 = [\pi_{h_1}, \pi_{h_1+1}, \dots, \pi_{h_1+l_1}]$  and  $U_2 = [\pi_{h_2}, \pi_{h_2+1}, \dots, \pi_{h_2+l_2}]$  are related, if they satisfy that*

- $\pi_{h_1+1} < \pi_{h_1+2}$
- $h_2 + l_2 + 1 = h_1$ , i.e. they appear consecutively in  $\pi$ ;
- $\pi_{h_2+j} < \pi_{h_1+i}$ , for all  $1 \leq i \leq l_1$  and  $1 \leq j \leq l_2$ ;
- $\pi_{h_2} > \pi_{h_1+1}$ .

we also say that  $U_2$  is related to  $U_1$  and the arc  $(\pi_{h_2}, \pi_{h_1+1})$  is called the related arc.

How do we identify the related umbrellas from the input permutation  $\pi$ ? We identify  $U_1$  such that  $\pi_{h_1}$  is the maximum element in  $\pi$ . Assume that we have identified  $U_i$ , if  $U_i$  satisfies one of the following conditions,

- condition (1):  $|E[U_i]| = 5$ , and  $\pi_{h_i+2} > \pi_{h_i+1}$  after converting  $U_i$  to be simple;
- condition (2):  $|E[U_i]| = 3$ , and  $\pi_{h_i+2} > \pi_{h_i+1}$ ;
- condition (3):  $|E[U_i]| = 1$ .

we scan  $\pi$  from right to left starting from  $\pi_{h_i}$ , the first element which is greater than  $\pi_{h_i+1}$  (if exists) is  $\pi_{h_{i+1}}$ ; otherwise we stop and  $U_1, \dots, U_i$  is a series of related umbrellas. So the last umbrella  $U_k$ , in a series of related umbrellas  $U_k, U_{k-1}, \dots, U_1$ , do not satisfy condition (1), (2), (3), then it satisfies one of the following conditions,

- condition (4):  $|E[U_k]|$  is even;
- condition (5):  $|E[U_k]|$  is odd, and  $|E[U_k]| \geq 6$ ;
- condition (6):  $|E[U_k]|$  is odd, and  $|E[U_k]| \leq 5$ , and  $\pi_{h_k+2} < \pi_{h_k+1}$ ;
- condition (7):  $|E[U_i]|$  is odd, and  $\pi_{h_i+1}$  is the minimum element in the minimal independent sub-permutation.

The algorithm for identifying related umbrellas is shown below.

Algorithm: *Identify-Related-Umbrella*( $\bullet$ )  
 Input:  $\pi = [\pi_1, \pi_2, \dots, \pi_n]$ , A preprocessed minimal independent sub-permutation.  
 Output: a series of related umbrellas  $\mu = [U_k, U_{k-1}, \dots, U_1]$ .

- 1  $k = 1, \pi_{h_k} = \max\{\pi_i | \pi_i \neq i, 1 \leq i \leq n\}, U_k = [\pi_{h_k}, \pi_{h_k+1}, \dots, \pi_n]$
- 2 While ( $U_k \neq \emptyset$ )
  - {
  - 3 if ( $U_k$  fulfils one of the conditions (1)(2)(3)){
  - 4 let  $0 < j < h_k$  and  $j = \min\{i | \pi_{h_k-i} > \pi_{h_k+1}\}$
  - 5 if ( $j$  exists){
  - 6  $k = k + 1, \pi_{h_k} = \pi_{h_{k-1}-j}, U_k = [\pi_{h_k}, \pi_{h_k+1}, \dots, \pi_{h_{k-1}+1}]$
  - 7 else break. }
  - 8 else break. }
  - }
- 9 return  $\mu = [U_k, U_{k-1}, \dots, U_1]$ .

**Lemma 9.** *Given a series of related umbrellas  $U_k, U_{k-1}, \dots, U_1$ , where  $U_i$  is related to  $U_{i-1}$  ( $2 \leq i \leq k$ ), if  $U_1, \dots, U_{k-1}$  are all odd and  $U_k$  is even ( resp. odd ), then there exists a group of Hops( resp. and one skip), which can remove all the inside-arcs of each umbrella, as well as the  $k - 1$  related arcs.*

The algorithm for sorting related umbrellas is shown below.

Algorithm: *Sort-Related-Umbrellas*( $\bullet$ )

- 1 for  $i = k$  down to 1{
- 2 Sort-Single-Umbrella( $U_i$ ),  $d = d + \lceil \frac{|E[U_i]|}{2} \rceil$ .
- 3 if ( $i > 1$ )
- 4 {  $\rho[d] = \rho(\pi_{h_i}, \pi_{h_{i-1}}, \pi_{h_{i-1}+1}), d = d + 1$
- 5  $U_{i-1} = [\pi_{h_{i-1}}, \pi_{h_i+2}, \dots, \pi_{h_i+l_i}]$  }
- 6  $\mu' = \mu \cdot \rho[1] \cdot \dots \cdot \rho[d]$
- 7 return  $\rho[0 \sim d - 1], \mu'$ .

### 5.4 Eliminate Crossing Arcs

Besides the inside-arcs and the related arcs, there may also be the following four types of arcs:

- type-1 (head-member crossing arcs): arcs from  $\pi_{h_i}$  to the members (except  $\pi_{h_{i-1}+1}$ ) of  $U_j$  ( $2 \leq j < i \leq k$ ).
- type-2 (head-head crossing arcs): arcs form  $\pi_{h_i}$  to  $\pi_{h_j}$  ( $2 \leq j < i \leq k$ ).
- type-3 (external-head arcs): arcs from the elements not in  $\mu$  to  $\pi_{h_i}$  ( $2 \leq i \leq k$ ).
- type-4 (external-member arcs): arcs from the elements not in  $\mu$  to the members of  $U_i$  ( $1 \leq i \leq k$ ).

Then, we try to remove the two types crossing arcs. Consider the scenario of the permutation  $\mu'$  after performing the algorithm *Sort-Related-Umbrella*( $\bullet$ ),

let  $re(U_i)$  be the sorted sub-permutation of the members of  $U_i$  except  $\pi_{h_{i+1}}$ , then  $\mu' =$

$$[\pi_{h_{k+1}}, re(U_k), \pi_{h_{k-1}+1}, \pi_{h_k}, re(U_{k-1}), \dots, re(U_2), \pi_{h_1+1}, \pi_{h_2}, re(U_1), \pi_{h_1}].(\star)$$

We now present some properties of  $\mu'$ . Firstly,  $\mu'$  would be in sorted order provided that we had deleted all the  $\pi_{h_i}$ 's ( $2 \leq i \leq k$ ). So we have,

*Property 2.* The head-member crossing arcs from  $\pi_{h_i}$  ( $2 \leq i \leq k$ ) to some elements of  $[re(U_{i-1}), \pi_{h_{i-2}+1}]$  form a claw-umbrella.

We use  $U'_i$  to denote this kind of new claw-umbrella, with  $\pi_{h_i}$  being the head. Note that  $U'_1 = \emptyset$ . Actually, the head-arcs of each  $U'_i$  are type-1 crossing arcs from  $\pi_{h_i}$  to the members (except  $\pi_{h_{i-1}+1}$ ) of  $U_{i-1}$  or from  $\pi_{h_i}$  to  $\pi_{h_{i-2}+1}$  ( $2 \leq i \leq k$ ).

*Property 3.* If  $L[U'_i] > 1$ , then  $L[U_{i-1}] \geq 3$ .

The  $U'_i$ 's may be also related, and the related arcs are type-1 head-member crossing arcs before performing the algorithm Sort-Related-Umbrella( $\bullet$ ), so there could be some minimal independent sub-permutations in  $\mu'$ , the permutation graph of such a minimal independent sub-permutation form a connected component. Let  $C_{j \rightarrow i}$  be the connected component from  $U'_j$  to  $U'_i$ . Then we have,

*Property 4.* Only the the connected component containing  $U'_k$  can have type-3,4 arcs, and if so, there must also be type-3,4 arcs to all the elements of  $[\pi_{h_{k+1}}, re(U_k), \pi_{h_{k-1}+1}]$ .

So while sorting  $\mu'$ , we remove arcs of the connected components from left to right respectively, provided that the connected component does not have type-3,4 arcs. For the connected component  $C_{j \rightarrow i}$ , we check each  $\pi_{h_r}$  as  $r$  increases from  $i$  to  $j$ , once we find an umbrella  $U_r''$  with  $\pi_{h_r}$  being the head, and there is no type-2 arc from other elements to  $\pi_{h_r}$ , then the umbrella  $U_r''$  can be sorted.

The algorithm for removing crossing arcs is shown below.

Algorithm: *Remove-Crossing-Arcs*( $\bullet$ )

- 1 for each connected component  $C_{j \rightarrow i}$ , not containing type-3,4 arcs. {
- 2     for  $r = i$  to  $j$
- 3         if (there is no type-2 arcs to  $\pi_{h_r}$ )
- 4             { Sort-Single-Umbrella( $U_r''$ ),  $d = d + \lceil \frac{|E[U_r'']|}{2} \rceil$ . }
- 5     return  $\rho[d \sim d + d']$ .

The connected component, containing type-3,4 arcs, is not handled here, and will be handled in our full algorithm. Our main algorithm is as follows.

In the algorithm Sort-Short-Block-Moves( $\bullet$ ), each iteration of the While-Loop is call a round. There are two stage in each round, the fist stage is *Sort-Related-Umbrellas*( $\bullet$ ), and the second stage is *Remove-Crossing-Arcs*( $\bullet$ ).

Algorithm: *Sort-Short-Block-Moves*( $\bullet$ )  
 Input:  $\pi$ , A preprocessed minimal independent sub-permutation.  
 Output: The short block move operations to sort  $\pi$ .

- 1 While (there exist  $j$  such that  $\pi_j \neq j$ ) {
- 2   Identify-Related-Umbrella( $\bullet$ ).
- 3   If( $U_k$  is odd and  $|E[U_k]| \leq 6$  and  $\pi_{h_k+1} > \pi_{h_k+2}$ )
- 4     {Apply Hop  $\rho(\pi_{h_k}, \pi_{h_k+1}, \pi_{h_k+2})$ ,  $U_k = [\pi_{h_k}, \pi_{h_k+3}, \dots, \pi_{h_k+l_k}]$  }.
- 5   If( $U_k$  is odd and  $|E[U_k]| \leq 6$  and  $\pi_{h_k+1} < \pi_{h_k+2}$ )
- 6     {Apply Skip  $\rho(\pi_{h_k}, \pi_{h_k+1})$ ,  $U_k = [\pi_{h_k}, \pi_{h_k+2}, \dots, \pi_{h_k+l_k}]$  }.
- 7   Sort-Related-Umbrellas( $\bullet$ ).
- 8   Remove-Crossing-Arcs( $\bullet$ ).

In each round, after the second stage, there may also be some arcs (the connected component containing type-3,4 arcs ) not removed. These arcs will flow into the next round.

## 6 The Approximation Factor Is 1.25

In this section, we prove that the algorithm *Sort-Short-Block-Moves*( $\bullet$ ) approximates the optimal solution within a factor  $5/4$ . Recall that, from Theorem 2, it is sufficient to prove that  $(H + S') / (S - S') \geq 3/2$ . We show that this inequality holds for each round of our algorithm. The following lemma is directly from the algorithm Sort-Related-Umbrellas( $\bullet$ ).

**Lemma 10.** *Given a series of related umbrellas  $U_k, U_{k-1}, \dots, U_1$ , where  $U_i$  is related to  $U_{i-1}$  ( $2 \leq i \leq k$ ), and  $U_i = [\pi_{h_i}, \pi_{h_i+1}, \dots, \pi_{h_i+l_i}]$  ( $1 \leq i \leq k$ ), there are  $k - 1$  Hops of the form  $\rho(\pi_{h_i}, \pi_{h_i-1}, \pi_{h_i-1+1})$ , which remove the related arcs and a head-arc of  $U_{i-1}$  ( $2 \leq i \leq k$ ), while performing the algorithm Sort-Related-Umbrellas( $\bullet$ ).*

We call the  $k - 1$  Hops of the form  $\rho(\pi_{h_i}, \pi_{h_i-1}, \pi_{h_i-1+1})$  existing-hops. The lemma followed is straightly from Lemma 8 and the algorithm Remove-Crossing-Arcs( $\bullet$ ).

**Lemma 11.** *Given a series of related umbrellas  $U_k, U_{k-1}, \dots, U_1$ , where  $U_i$  is related to  $U_{i-1}$  ( $2 \leq i \leq k$ ), and  $U_i = [\pi_{h_i}, \pi_{h_i+1}, \dots, \pi_{h_i+l_i}]$  ( $1 \leq i \leq k$ ), there are at most  $k - 1$  Skips of the form  $\rho(\pi_{h_i}, *)$ , which remove a cross edge of  $\pi_{h_i}$  ( $2 \leq i \leq k$ ), while perform the algorithm Remove-Crossing-Arcs( $\bullet$ ).*

Further, we analyze the number of Hops and Skips while performing the algorithm Remove-Crossing-Arcs( $\bullet$ ).

**Lemma 12.** *Given a series of related umbrellas  $U_k, U_{k-1}, \dots, U_1$ , where  $U_i$  is related to  $U_{i-1}$  ( $2 \leq i \leq k$ ), and  $U_i = [\pi_{h_i}, \pi_{h_i+1}, \dots, \pi_{h_i+l_i}]$  ( $1 \leq i \leq k$ ). If  $|E[U_i]| \leq 3$ ,  $|E[U_{i-1}]| \leq 3$  and  $|E[U_{i-2}]| \leq 3$  then  $U'_i$  and  $U'_{i-1}$  can not be both lone-arc, unless there exists a necessary Skip to remove  $U'_i$  and  $U'_{i-1}$ , for  $3 \leq i \leq k$ .*

*Proof.* Since each umbrella of  $U_{k-1}, \dots, U_1$  fulfills one of conditions (1),(2),(3), if  $|E[U_i]| \leq 3$ ,  $|E[U_{i-1}]| \leq 3$  and  $|E[U_{i-2}]| \leq 3$ , then they are 3-claws or 1-claws, which means  $[U'_i, U'_{i-1}, U'_{i-2}]$  is a double increasing sub-permutation. In the  $(\star)$  formula, if  $U'_i$  is a lone-arc, it is either  $(\pi_{h_i}, \pi_{h_{i-2}+1})$  ( which implies  $|E[U_{i-1}]| = 1$ ) or  $(\pi_{h_i}, \pi_{h_{i-1}+r})$  ( where  $\pi_{h_{i-1}+r}$  is the second minimum element in  $U_{i-1}$  ). This statement also holds for  $U'_{i-1}$ . By enumerating the four possible cases, we can prove that there exists a necessary Skip.

**Corollary 2.** *For any pair of umbrellas  $U_i$  and  $U_{i-1}$  ( $3 \leq i \leq k$ ), one of the three cases holds: (I) at least one of  $U'_i$  and  $U'_{i-1}$  is not lone arc; (II) there exists a necessary Skip to remove  $U'_i$  and  $U'_{i-1}$ ; (III) at least one of  $U_i, U_{i-1}$  and  $U_{i-2}$  has at least five inside-arcs.*

Note that case (I) implies that there is at least one Hop while remove  $U'_i$  and  $U'_{i-1}$  in the algorithm Remove-Crossing-Arcs( $\bullet$ ); case(III) implies that there are at least two Hops while sort the umbrella with five edges in the algorithm Sort-Related-Umbrellas( $\bullet$ ). So, case(I) is worst.

In the  $t$ 'th round of the algorithm Sort-Short-Block-Moves( $\bullet$ ), let  $\mu=[U_k^t, U_{k-1}^t, \dots, U_1^t]$  be the related umbrellas, where  $U_i^t$  is related to  $U_{i-1}^t$  ( $2 \leq i \leq k$ ), and  $U_i^t = [\pi_{h_i}^t, \pi_{h_i+1}^t, \dots, \pi_{h_i+l_i}^t]$  ( $1 \leq i \leq k$ ). Let  $U_j^{tt}$  ( $2 \leq j \leq k$ ) be the last umbrella handled by the algorithm Remove-Crossing-Arcs( $\bullet$ ). Let  $H, S, S'$  be the number of Hops, Skips and necessary Skips respectively, while performing the algorithms Sort-Related-Umbrellas( $\bullet$ ) and Remove-Crossing-Arcs( $\bullet$ ). According to the condition of  $U_k^t$ , we have the following two lemmas.

**Lemma 13.** *If  $U_k^t$  satisfies one of conditions (4),(5),(6), then,  $(H + S' - (k - j))/(S - S' - (k - j)) \geq 3/2$ .*

*Proof.* From Lemma 10,  $H \geq k - 1$ , including the  $j - 1$  existing-hops of the form  $\rho(\pi_{h_i}^t \pi_{h_{i-1}}^t, \pi_{h_{i-1}+1}^t)$  ( $2 \leq i \leq j$ ). From Lemma 11,  $S \leq k - 1$ , including the skips of  $\pi_{h_i}^t$  ( $2 \leq i \leq j$ ). If  $U_2^{tt}$  is a lone-arc,  $|E[U_1^t]| \geq 3$ , which means there is at least one Hop while sorting  $U_1^t$ . Consider the contribution of two consecutive umbrellas  $U_i^t$  and  $U_{i-1}^t$  ( $2 \leq i \leq j - 1$ ) for each case in Corollary 2. If it is case (I), the numerator would be added by one; if it is case (II), the numerator would be added by one, while the denominator would be subtracted by one; if it is case (III), the numerator would be added by two.

If  $U_2^{tt}$  is a lone-arc, there are at least  $\lfloor (j-2)/2 \rfloor$  more Hops ( the contribution of  $U_2^t, \dots, U_{j-1}^t$  ), so we have

$$(H + S' - (k - j))/(S - S' - (k - j)) \geq (j + \lfloor (j - 2)/2 \rfloor)/(j - 1) \geq 3/2.$$

If  $U_2^{tt}$  is not a lone-arc, there are at least  $\lfloor (j - 3)/2 \rfloor$  more Hops ( the contribution of  $U_3^t, \dots, U_{j-1}^t$  ), so we have

$$(H + S' - (k - j))/(S - S') \geq (j - 1 + \lfloor (j - 3)/2 \rfloor)/(j - 2) \geq 3/2.$$

□

It remains to consider the short block moves to remove the type-3 external-head arcs and type-4 external-member arcs. We handle the two possibilities:  $k = j$  or not respectively. The details are not shown here due to space limitation.

Due to our algorithm Identify-Related-Umbrella(●), only the last umbrella of the last round could satisfy condition (7), and there are no type-4 external-member arcs in the last round. Let  $\mu = [U_k^m, U_{k-1}^m, \dots, U_1^m]$  be the related umbrellas of the last round, where  $U_i^m$  is related to  $U_{i-1}^m$  ( $2 \leq i \leq k$ ), and  $U_i^m = [\pi_{h_i}^m, \pi_{h_i+1}^m, \dots, \pi_{h_i+l_i}^m]$  ( $1 \leq i \leq k$ ). Let  $H, S, S'$  be the number of Hops, Skips and necessary Skips respectively, while performing the algorithms Sort-Related-Umbrellas(●) and Remove-Crossing-Arcs(●).

**Lemma 14.** *If  $U_k^t$  satisfies condition (7), then  $(H + S')/(S - S') \geq 3/2$ .*

*Proof.* Since  $U_k^m$  can be converted into an even umbrella by performing the Skip  $\rho(\pi_{h_k}^m, \pi_{h_k+1}^m)$ , if  $U_k^m$  is not a lone arc, according to Lemma 13, we are done. If  $U_k^m$  does be a lone arc, we prove this lemma by analyzing the size of  $E[U_k^m]$ .

If  $|E[U_k^m]| \leq 3$ , i.e.,  $U_k^m$  is a 1-claw or a 3-claw, then  $|E[U_{k-1}^m]| = 5$ ; otherwise, according to Lemma 5, there exists a necessary Skip. Consider the contribution of two consecutive umbrellas  $U_i^m$  and  $U_{i-1}^m$  ( $2 \leq i \leq k - 2$ ) for each case in Corollary 2. If  $U_2^t$  is a lone-arc, there are at least  $\lfloor (k - 3)/2 \rfloor$  more Hops, so we have

$$(H + S')/(S - S') \geq (k - 1 + 1 + 2 + \lfloor (k - 3)/2 \rfloor)/(k - 1 + 1) \geq 3/2.$$

If  $U_2^t$  is not a lone-arc, there are at least  $\lfloor (k - 4)/2 \rfloor$  more Hops, so we have

$$(H + S')/(S - S') \geq (k - 1 + 2 + \lfloor (k - 4)/2 \rfloor)/(k - 2 + 1) \geq 3/2.$$

If  $|E[U_k^m]| \geq 5$ , then there are at least two Hops while sorting  $U_k^m$ . Consider the contribution of two consecutive umbrellas  $U_i^m$  and  $U_{i-1}^m$  ( $2 \leq i \leq k - 1$ ) for each case in Corollary 2.

If  $U_2^t$  is a lone-arc, there are at least  $\lfloor (k - 2)/2 \rfloor$  more Hops, so we have

$$(H + S')/(S - S') \geq (k - 1 + 1 + 2 + \lfloor (k - 2)/2 \rfloor)/(k - 1 + 1) \geq 3/2.$$

If  $U_2^t$  is not a lone-arc, there are at least  $\lfloor (k - 3)/2 \rfloor$  more Hops, so we have

$$(H + S')/(S - S') \geq (k - 1 + 2 + \lfloor (k - 3)/2 \rfloor)/(k - 2 + 1) \geq 3/2.$$

□

**Theorem 3.** *The algorithm Sort-Short-Block-Moves(●) approximates the short block move distance with a factor 1.25.*

**Acknowledgments.** This research is partially supported NSF of China under grant 61202014 and 61472222, by NSF of Shandong Province ( China ) under grant ZR2012FQ008 and ZR2012FZ002, by China Postdoctoral Science Foundation funded project under grant 2011M501133 and 2012T50614.



## References

1. Bafna, V., Pevzner, P.: Sorting by transpositions. *SIAM Journal on Discrete Mathematics* **11**(2), 224–240 (1998)
2. Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) *ICALP 2011, Part I. LNCS*, vol. 6755, pp. 654–665. Springer, Heidelberg (2011)
3. Elias, I., Hartman, T.: A  $1375$ -approximation algorithm for sorting by transpositions. *IEEE/ACM Trans. Comput. Biology Bioinform* **3**(4), 369–379 (2006)
4. Eriksson, H., Eriksson, K., Karlander, J., Svensson, L., Wätlund, J.: Sorting a bridge hand. *Discrete Mathematics* **241**(1–3), 289–300 (2001)
5. Hall, P.: On Representatives of Subsets. *J. London Math. Soc.* **10**, 26–30 (1935)
6. Heath, L.S., Vergara, J.P.C.: Sorting by bounded blockmoves. *Discrete Applied Mathematics* **88**, 181–206 (1998)
7. Heath, L.S., Vergara, J.P.C.: Sorting by short block moves. *Algorithmica* **28**(3), 323–354 (2000)
8. Jiang, H., Zhu, D., Zhu, B.: A  $(1+\epsilon)$ -approximation algorithm for sorting by short block-moves. *Theoretical Computer Science* **54**(2), 279–292 (2011)
9. Jiang, H., Zhu, D.: A  $14/11$ -approximation algorithm for sorting by short block-moves. *Science China Information Sciences* **54**(2), 279–292 (2011)
10. Mahajan, M., Rama, R., Vijayakumar, S.: On sorting by  $3$ -bounded transpositions. *Discrete Mathematics* **306**(14), 1569–1585 (2006)
11. Sankoff, D., Leduc, G., Antoine, N., Paquin, B., Lang, B.F., Cedergren, R.: Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. Nat. Acad. Sci. USA* **89**, 6575–6579 (1992)

# **Online and Approximation Algorithms**

# Lower Bounds for On-line Graph Colorings

Grzegorz Gutowski<sup>1</sup>, Jakub Kozik<sup>1</sup>, Piotr Micek<sup>1</sup> (✉), and Xuding Zhu<sup>2,3</sup>

<sup>1</sup> Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Krakow, Poland

{gutowski,jkozik,micek}@tcs.uj.edu.pl

<sup>2</sup> Department of Mathematics, Zhejiang Normal University, Jinhua, China

<sup>3</sup> Department of Applied Mathematics, National Sun Yat-sen University, Kaohsiung, Taiwan  
xdzhu@zjnu.edu.cn

**Abstract.** We propose two strategies for Presenter in on-line graph coloring games. The first one constructs bipartite graphs and forces any on-line coloring algorithm to use  $2\log_2 n - 10$  colors, where  $n$  is the number of vertices in the constructed graph. This is best possible up to an additive constant. The second strategy constructs graphs that contain neither  $C_3$  nor  $C_5$  as a subgraph and forces  $\Omega(\frac{n}{\log n^{\frac{1}{3}}})$  colors. The best known on-line coloring algorithm for these graphs uses  $O(n^{\frac{1}{2}})$  colors.

## 1 Introduction

A *proper coloring* of a graph  $G$  is an assignment of colors to the vertices of the graph such that adjacent vertices receive distinct colors. An  $n$ -round *on-line coloring game* on a class of graphs  $\mathcal{G}$  is a two-person game, played by Presenter and Algorithm. In each round Presenter introduces a new vertex of a graph with its adjacency status to all vertices presented earlier. The only restriction for Presenter is that in every moment the currently presented graph is in  $\mathcal{G}$ . Algorithm assigns colors to the incoming vertices in such a way that the coloring of the presented graph is always proper. The color for a new vertex has to be assigned before Presenter introduces the next vertex. The assignment is irrevocable. The goal of Algorithm is to minimize the number of different colors used during the game.

Throughout the paper  $\log$  and  $\ln$  are logarithm functions to base 2 and  $e$ , respectively. By the *size* of a graph we mean the number of vertices in the graph.

For most classes of graphs the number of colors necessary in the corresponding on-line coloring game can not be bounded in terms of the chromatic number of the constructed graph. Rare examples of classes where it is possible include interval graphs [8], more generally cocomparability graphs [6] or  $P_5$ -free graphs [7]. All of these results are covered by the main result of [6] that says that for any tree  $T$  with radius 2, the class of graphs that do not contain an

---

This research is supported by: Polish National Science Center UMO-2011/03/D/ST6/01370.

induced copy of  $T$  can be colored on-line with number of colors being a function of  $T$  and chromatic number of presented graph.

Usually, for general enough classes of graphs, the best one can hope for is to bound the number of colors used in an on-line coloring game in terms of the number of rounds (i.e. the size of the constructed graph).

It is a popular exercise to show a strategy for Presenter that constructs forests of size  $n$  and forces Algorithm to use at least  $\lfloor \log n \rfloor + 1$  colors. On the other hand, *First-Fit strategy* for Algorithm (that is a strategy that colors each incoming vertex with the least admissible natural number), uses at most  $\lfloor \log n \rfloor + 1$  colors on forests of size  $n$ .

When the game is played on bipartite graphs, Presenter can easily trick out First-Fit strategy and force  $\lceil \frac{n}{2} \rceil$  colors on a bipartite graph of size  $n$ . Lovász, Saks and Trotter [11] gave a simple strategy for Algorithm using at most  $2 \log n$  colors on bipartite graphs of size  $n$ . Recently, Bianchi et al. [2] proposed a strategy for Presenter that forces Algorithm to use at least  $\lceil 1.13746 \log n - 0.49887 \rceil$  colors on bipartite graphs of size  $n$ . We improve this bound to  $2 \log n - 10$ , which matches an upper bound from [11] up to an additive constant.

**Theorem 1.** *There exists a strategy for Presenter that forces at least  $2 \log_2 n - 10$  colors on bipartite graphs of size  $n$ .*

Perhaps, the most exciting open problem in the area is whether there is a strategy for Algorithm using  $O(n^{1-\varepsilon})$  colors on triangle-free graphs of size  $n$ , for some  $\varepsilon > 0$ . The only non-trivial on-line algorithm for triangle-free graphs, given by Lovász et al. [11], uses  $O(\frac{n}{\log \log n})$  colors on graphs of size  $n$ . Triangle-free graphs may have arbitrarily large chromatic number, but the chromatic number in this case has a precise bound in terms of the number of vertices. Ajtai, Komlós and Szemerédi [1] proved that the chromatic number of any triangle-free graph of size  $n$  is  $O\left(\frac{n}{\log n}^{\frac{1}{2}}\right)$ . Kim [9] presented a probabilistic construction of triangle-free graphs with chromatic number  $\Omega\left(\frac{n}{\log n}^{\frac{1}{2}}\right)$ .

The *girth* of a graph  $G$  is the length of the shortest cycle in  $G$ . The *odd-girth* of  $G$  is the length of the shortest odd cycle in  $G$ . Thus, triangle-free graphs have odd-girth at least 5. For bipartite graphs the odd-girth is not defined and for convenience it is set to  $\infty$ . For graphs with odd-girth at least 7, i.e.  $C_3$ - and  $C_5$ -free, Kierstead [5] gave a strategy for Algorithm that uses  $O\left(n^{\frac{1}{2}}\right)$  colors on such graphs of size  $n$ . Curiously, no better strategy for Algorithm is known even for classes of graphs with odd-girth larger than any  $g \geq 7$ . On the off-line side, Denley [3] has shown that the chromatic number of graphs of size  $n$  with odd-girth at least  $g \geq 7$  is  $O\left(\frac{n}{\log n}^{\frac{2}{g-1}}\right)$ . For the lower bound, it is well-known (see Lemma 6.1 in Krivelevich [10]) that there are graphs of size  $n$  and with girth at least  $g$  with chromatic number  $\Omega\left(n^{\frac{1}{g-2}}\right)$ .

For a good introduction to our second result, we present a simple strategy for Presenter by Diwan, Kenkre and Vishwanathan [4] that forces Algorithm to use  $\Omega\left(n^{\frac{1}{2}}\right)$  colors on triangle-free graphs of size  $n$ . Note that it improves the

bound that trivially follows from the non-trivial off-line construction only by a logarithmic factor. For the case of graphs with odd-girth at least 7 we propose a strategy for Presenter that forces Algorithm to use  $\Omega\left(\frac{n}{\log n}^{\frac{1}{3}}\right)$  colors.

**Theorem 2.** *There exists a strategy for Presenter that forces  $\Omega\left(\frac{n}{\log n}^{\frac{1}{3}}\right)$  colors on graphs of size  $n$  with odd-girth at least 7.*

## 2 Bipartite Graphs

*Proof (Proof of Theorem 1).* We give a strategy for Presenter that forces Algorithm to use  $c$  different colors on bipartite graphs of size  $(8 + 7\sqrt{2})2^{\frac{c}{2}}$ . Thus, Presenter can force  $\lfloor 2 \log n - 2 \log(8 + 7\sqrt{2}) \rfloor \geq \lfloor 2 \log n - 8.32 \rfloor$  colors on bipartite graphs of size  $n$ .

At any moment during the game the presented graph is bipartite and consists of a number of connected components. Each component has the unique bipartition into two independent sets which we call the *sides* of the component. A color  $\alpha$  is *one-sided* in a component  $C$  if there is a vertex in  $C$  colored with  $\alpha$  but only in one out of the two sides of  $C$ . A color  $\alpha$  is *two-sided* in  $C$  if there are vertices in both sides of  $C$  colored with  $\alpha$ . The set of two-sided colors in a component  $C$  is denoted by  $ts(C)$ . The *level* of a component  $C$ , denoted by  $lev(C)$ , is the number of two-sided colors in  $C$ .

The strategy is divided into phases in which carefully chosen components are merged or a new component being a single edge is introduced. After each phase, for each presented component  $C$  the strategy maintains two *selected* vertices in opposite sides of  $C$  colored with one-sided colors. The two-element set of colors assigned to the selected vertices of a component  $C$  is denoted by  $sel(C)$ .

Consider a single phase of the strategy. If a new component  $C$ , which is always a single edge, is introduced then  $C$  has no two-sided colors and both vertices of  $C$  are selected. If the strategy *merges* components  $C_1, \dots, C_k$ , all with the same level, then for every  $C_i$  the strategy fixes one side of  $C_i$  to be the left side, and the other to be the right side. After that, two adjacent vertices  $a$  and  $b$  are introduced. The vertex  $a$  is adjacent to all the vertices in the left sides of components  $C_1, \dots, C_k$  and  $b$  is adjacent to all the vertices in the right sides. Let  $C$  be the component created by this merge. The colors assigned to  $a$  and  $b$  are one-sided colors in  $C$  and the strategy chooses  $a$  and  $b$  to be selected for  $C$ . Observe that  $ts(C) = \bigcup_{i=1}^k ts(C_i) \cup X$ , where a color  $\alpha$  is in  $X$  if there are different  $C_i$  and  $C_j$  such that  $\alpha$  is one-sided in both  $C_i$  and  $C_j$ ,  $\alpha$  appears on a vertex in the left side of  $C_i$  and on a vertex in the right side of  $C_j$ .

A single phase is described by the following four rules. For each phase the strategy uses the first applicable rule.

- (1) **Merge Different.** If there are two components  $C_1, C_2$  with the same level and  $|ts(C_1) \setminus ts(C_2)| \geq 2$ , then merge those two components into a new component  $C$ . Note that  $lev(C) \geq lev(C_1) + 2$  and  $|C| = |C_1| + |C_2| + 2$ .

- (2) **Merge Similar.** If there are two components  $C_1, C_2$  with the same level and  $|\text{ts}(C_1) \setminus \text{ts}(C_2)| = 1$  and  $\text{sel}(C_1) \cap \text{sel}(C_2) \neq \emptyset$ , then merge those two components into a new component  $C$  in such a way that a common one-sided color becomes two-sided in  $C$ . Note that  $\text{lev}(C) \geq \text{lev}(C_1) + 2$  and  $|C| = |C_1| + |C_2| + 2$ .
- (3) **Merge Equal.** If there are  $k \geq 2$  components  $C_1, \dots, C_k$  with the same level and  $\text{ts}(C_1) = \dots = \text{ts}(C_k)$  and there are  $k$  distinct colors  $\alpha_1, \dots, \alpha_k$  such that  $\text{sel}(C_1) = \{\alpha_1, \alpha_2\}$ ,  $\text{sel}(C_2) = \{\alpha_2, \alpha_3\}, \dots, \text{sel}(C_k) = \{\alpha_k, \alpha_1\}$ , then merge those  $k$  components into a new component  $C$  in such a way that each of the  $\alpha_i$ 's becomes two-sided in  $C$ . Note that  $\text{lev}(C) \geq \text{lev}(C_1) + k$  and  $|C| = \sum_{i=1}^k |C_i| + 2$ .
- (4) **Introduce.** Introduce a new component  $C$  being a single edge. Note that  $\text{lev}(C) = 0$  and  $|C| = 2$ .

This concludes the description of a single phase of the strategy. Now we present two simple invariants kept by the strategy.

**Invariant 1.**  $|C| \leq 2^{\frac{\text{lev}(C)}{2} + 2} - 2$  for every component  $C$  after each phase of the strategy.

The statement vacuously holds at the beginning of the on-line game. We show that the invariant holds from phase to phase. Clearly, it suffices to argue that a component  $C$  created in a considered phase satisfies the statement. If  $C$  is a product of  $k \geq 2$  components  $C_1, \dots, C_k$  by one of the merging rules (1)-(3) then

$$\begin{aligned}
 |C| &= \sum_{i=1}^k |C_i| + 2 \leq \sum_{i=1}^k \left( 2^{\frac{\text{lev}(C_i)}{2} + 2} - 2 \right) + 2 \\
 &\leq k \cdot 2^{\frac{\text{lev}(C) - k}{2} + 2} - 2(k - 1) = 2^{\frac{\text{lev}(C)}{2} - \frac{k}{2} + \log k + 2} - 2(k - 1) \\
 &\leq 2^{\frac{\text{lev}(C)}{2} + 2} - 2.
 \end{aligned}$$

If  $C$  is a component introduced by Rule (4), then invariant holds trivially.

**Invariant 2.** After each phase, none of the merging rules (1)-(3) applies to the set of all available components without the component created in the last phase.

The statement trivially holds after the first phase. Let  $C_k$  be the component created in the  $k$ -th phase. If  $C_k$  is introduced by Rule (4), then rules (1)-(3) do not apply in the graph without  $C_k$ . If  $C_k$  is merged from some components available after phase  $k - 1$ , then by induction hypothesis  $C_{k-1}$  is one of the merged components. Thus, the set of components after phase  $k$  without  $C_k$  is a subset of the set of components after phase  $k - 1$  without  $C_{k-1}$ . Therefore, the statement follows by induction hypothesis.

For the further analysis we need one more observation. Suppose that Rule (3) does not apply to the current set of components and there is a set of colors  $T$  and  $p$  distinct components  $C_1, \dots, C_p$  with  $\text{ts}(C_i) = T$  for all  $i \in \{1, \dots, p\}$ . We claim that  $|\bigcup_{i=1}^p \text{sel}(C_i)| \geq p + 1$ . Indeed, consider a multigraph  $M$  with vertex set  $\bigcup_{i=1}^p \text{sel}(C_i)$  and  $p$  edges formed by all pairs of colors on selected vertices in

the components (i.e. for each  $C_i$  there is an edge connecting colors in  $\text{sel}(C_i)$ ). If  $|\bigcup_{i=1}^p \text{sel}(C_i)| \leq p$  then there is a cycle in  $M$ . Say that a cycle is defined by  $q \geq 2$  edges originating from components  $C_{i_1}, \dots, C_{i_q}$ . Then Rule (3) can be applied to components  $C_{i_1}, \dots, C_{i_q}$ , contradicting our assumption.

Fix  $c \geq 3$  and consider the situation in a game after a number of phases. Suppose that Algorithm used so far fewer than  $c$  colors. We are going to argue that Presenter introduced fewer than  $(8 + 7\sqrt{2})2^{\frac{c}{2}}$  vertices and this will conclude the proof. Clearly, for any available component  $C$  we have  $\text{lev}(C) \leq c - 3$  and therefore by Invariant 1 we have  $|C| \leq 2^{\frac{c-3}{2}+2} - 2$ . Let  $\mathcal{C}$  be the set of all components but the one created in the last phase. By Invariant 2 none of the merging rules (1)-(3) applies to  $\mathcal{C}$ .

Fix  $\ell \in \{0, \dots, c-3\}$  and let  $\mathcal{C}_\ell$  be the set of  $C \in \mathcal{C}$  with  $\text{lev}(C) = \ell$ . Now, we want to bound the number of components in  $\mathcal{C}_\ell$ . Let  $\{T_1, \dots, T_m\}$  be the set of values of  $\text{ts}(C)$  attained for  $C \in \mathcal{C}_\ell$ . Let  $p_i$  be the number of components  $C \in \mathcal{C}_\ell$  with  $\text{ts}(C) = T_i$  and let  $S_i = \bigcup_{\text{ts}(C)=T_i} \text{sel}(C)$ . As Rule (1) can not be applied  $|T_i - T_j| = 1$  for all distinct  $i, j \in \{1, \dots, m\}$ . In particular,  $|T_1 \cap S_i| \leq 1$  for all  $i \in \{2, \dots, m\}$ . As Rule (2) can not be applied  $S_i \cap S_j = \emptyset$  for all distinct  $i, j \in \{1, \dots, m\}$ . As Rule (3) can not be applied  $|S_i| \geq p_i + 1$  for all  $i \in \{1, \dots, m\}$ . All this give a lower bound on the number of colors in  $\bigcup_{i=1}^m S_i \cup T_1$ .

$$\begin{aligned} c - 1 &\geq \left| \bigcup_{i=1}^m S_i \cup T_1 \right| = \sum_{i=1}^m |S_i| + |T_1| - \sum_{i=1}^m |T_1 - S_i| \\ &\geq \sum_{i=1}^m (p_i + 1) + \ell - (m - 1). \end{aligned}$$

Thus,  $|\mathcal{C}_\ell| = \sum_{i=1}^m p_i \leq c - \ell - 2$ .

We define an auxiliary function  $f(i) = \sum_{j=0}^n (i - j)2^{\frac{j}{2}}$  and observe an easy bound  $f(i) < (4 + 3\sqrt{2})2^{\frac{i}{2}}$ . Using Invariant 1 and the bound on the number of components with any particular level, we get the following bound on the total number of vertices within components in  $\mathcal{C}$ :

$$\begin{aligned} \sum_{\ell=0}^{c-3} |\mathcal{C}_\ell| \cdot \left(2^{\frac{\ell}{2}+2} - 2\right) &\leq \sum_{\ell=0}^{c-3} (c - \ell - 2) \cdot \left(2^{\frac{\ell}{2}+2} - 2\right) \\ &< \sum_{\ell=0}^{c-2} (c - 2 - \ell) \cdot 4 \cdot 2^{\frac{\ell}{2}} = 4f(c - 2). \end{aligned}$$

To finish the proof we sum up the upper bounds on the size of the last produced component and the total size of all remaining components, that is

$$2^{\frac{c-3}{2}+2} - 2 + 4(4 + 3\sqrt{2})2^{\frac{c-2}{2}} < (8 + 7\sqrt{2})2^{\frac{c}{2}}.$$

□

### 3 The Odd-Girth

In this section we consider classes of graphs with odd-girth bounded from below. The high value of odd-girth of a graph implies that the graph is locally bipartite. However, it seems hard to exploit this property in an on-line framework. The only known on-line algorithms, that can use large odd-girth, is the one by Lovász et al. [11] using  $O(\frac{n}{\log \log n})$  colors for graphs of girth at least 4 and of size  $n$ , and the one by Kierstead [5] using  $O(n^{\frac{1}{2}})$  colors on graphs of size  $n$  with odd-girth at least 7. We present constructions that prove lower bounds for these problems for odd-girth at least 5 and at least 7.

**Theorem 3 (Diwan, Kenkre, Vishwanathan [4]).** *There exists a strategy for Presenter that forces  $\Omega(n^{\frac{1}{2}})$  colors on triangle-free graphs of size  $n$ .*

*Proof.* We give a strategy for Presenter that forces Algorithm to use  $c$  different colors. An auxiliary structure used by Presenter during the game is a table with  $c$  rows and  $c$  columns. Each cell of the table is initially empty, but over the time Presenter puts vertices into the table. There will be at most one vertex in each cell. Each time Algorithm colors a vertex with color  $i$ , the vertex is put into the last empty cell in the  $i$ -th row, e.g. the first vertex colored with  $i$  ends up in the cell in the  $i$ -th row and  $c$ -th column. Anytime Algorithm uses  $c$  different colors Presenter succeeds and the construction is complete.

The strategy is divided into  $c$  phases numbered from 0 to  $c - 1$ . Let  $I_0 = \emptyset$  and for  $k > 0$  let  $I_k$  be the set of vertices in the  $k$ -th column at the beginning of phase  $k$ . Now, as long as there is no vertex in the  $(k + 1)$ -th column with a color different than all the colors of the vertices in  $I_k$ , Presenter introduces new vertices adjacent to all vertices in  $I_k$  (and no other). The phase ends when some vertex  $v$  is put in the  $(k + 1)$ -th column. Clearly, each phase will end as Algorithm must use colors different than colors used on  $I_k$  for all vertices presented in the  $k$ -th phase and the space in the table where these vertices are stored is limited. Observe also, that the vertices presented in the  $k$ -th phase are never put into the  $k$ -th column. Hence, each edge connects vertices in different columns and each column in the table forms an independent set. As Presenter introduces only vertices with neighborhood contained within one column the constructed graph is triangle-free. Observe also that  $I_{k+1}$  is strictly larger than  $I_k$  and therefore  $|I_k| \geq k$ . Thus, after the completion of the  $(c - 1)$ -th phase each cell of the  $c$ -th column is filled and therefore Algorithm has already used  $c$  colors. As there are only  $c^2$  cells in the table, Presenter introduced at most that many vertices.  $\square$

Note that when Algorithm uses First-Fit strategy, the graphs constructed by Presenter are exactly shift graphs (a well known class of triangle-free graphs with chromatic number logarithmic in terms of their size). In the same vein the next strategy, for graphs with odd-girth at least 7, is inspired by the construction of double-shift graphs.

*Proof (Proof of Theorem 2).* We describe a strategy for Presenter that forces Algorithm to use  $c$  different colors. This time an auxiliary structure used by



Presenter is a table with  $c$  rows and  $3c$  columns. Each cell of the table is initially empty, but over the time Presenter puts vertices into the table. There will be at most  $3c$  vertices in each cell. All vertices put in the  $i$ -th row will be of color  $i$ . Every vertex in the table will have assigned a non-negative weight. If at some point Algorithm uses  $c$  different colors, Presenter succeeds and the construction is complete.

Color  $i$  is *available* for the  $j$ -th column, or the cell in the  $i$ -th row and  $j$ -th column is available, when there are fewer than  $3c$  vertices in this cell. Otherwise, the color, or the cell is *blocked*.

The strategy is divided into  $3c$  phases numbered from 0 to  $3c - 1$ . In the  $k$ -th phase, Presenter selects  $3c$  groups of vertices that are already in the table. If  $k = 0$  or there are no blocked cells in the  $k$ -th column then all the groups are empty. Otherwise, Presenter splits vertices of blocked colors for the  $k$ -th column in such a way that each group contains a vertex of each blocked color and no other vertices. Now for each group  $R$ , Presenter plays according to the following rules.

**Rule 1:** If there is a color available for the  $k$ -th column but blocked for all the columns to the right of the  $k$ -th column (i.e. for columns  $k + 1, \dots, 3c - 1$ ), then the whole phase is finished and Presenter starts the next phase. Phases that ended for this reason are called *broken*. Clearly, the number of broken phases is bounded by the number of colors, that is by  $c$ .

Otherwise, Presenter introduces an independent set  $F$  of  $3c(1 + \lceil \ln 3c \rceil)$  new vertices adjacent to all the vertices in  $R$ . Set  $F$  is called a *fan*. Now, Presenter investigates the possibilities of putting some of the vertices in  $F$  into the table but he restricts himself to put them only into one column which is to the right of the  $k$ -th column. We are going to use this property in the proof that the constructed graph contains neither  $C_3$  nor  $C_5$  as a subgraph.

**Rule 2:** If there is a cell in the table in a column to the right of the  $k$ -th column, say the cell in the  $i$ -th row and  $j$ -th column, such that there are  $m < 3c$  vertices in the cell (in particular the cell is available) and there are at least  $3c - m$  vertices in  $F$  colored by Algorithm with  $i$ , then Presenter puts  $3c - m$  vertices in  $F$  colored with  $i$  into the cell. From now on this cell is blocked. All vertices put into the table by this rule receive weight 0. All the other vertices in  $F$  (with color different than  $i$ ) are *discarded* and will not be used as neighbors for vertices introduced in the future.

**Rule 3:** If there is no such cell (i.e. we cannot apply Rule 2), then we call  $F$  an *interesting fan*. Let  $t_i$ , for  $i \in \{1, \dots, c\}$  be the number of columns to the right of the  $k$ -th column for which color  $i$  is available. Consider a bipartite graph with one part formed by  $3c - 1 - k$  columns to the right of the  $k$ -th column and the second part formed by  $3c(1 + \lceil \ln 3c \rceil)$  vertices in  $F$ . We put an edge in the graph between the  $j$ -th column and a vertex  $v \in F$  of color  $i$  if color  $i$  is available for the  $j$ -th column. To vertex  $v \in F$  of color  $i$  we assigned weight  $\frac{3c}{t_i}$ . All its incident edges also get weight  $\frac{3c}{t_i}$ . The total weight of all the edges in the graph is

$$\sum_{i \in \{1, \dots, c\}} \sum_{\substack{v \in F \\ v \text{ is colored with } i}} t_i \cdot \frac{3c}{t_i} = |F| \cdot 3c.$$

This means that for some column  $j > k$ , the total weight of incident edges is at least  $|F| \cdot \frac{3c}{3c-1-k} \geq |F|$ . Presenter puts all the vertices in  $F$  colored with available colors for the  $j$ -th column into the  $j$ -th column. The remaining vertices in  $F$  are discarded. Note that after this step all the cells in the  $j$ -th column still have at most  $3c$  vertices as otherwise Presenter would use Rule 2. Note also that the total weight assigned to the vertices put into the table is at least  $|F|$ . This finishes the description of the strategy for Presenter.

Note that there are at most  $3c$  phases of the game, and that in every phase at most  $3c$  fans of size  $3c(1 + \lceil \ln 3c \rceil)$  each, are presented. This gives no more than  $27c^3(1 + \lceil \ln 3c \rceil)$  vertices in total. We claim that before the end of the last phase Algorithm has to use  $c$  different colors. Suppose it does not. At most  $c$  of the phases are broken. During remaining phases Presenter introduces at least  $2c \cdot 3c$  fans. As there are  $c \cdot 3c$  cells in the table, and each cell can be blocked only once, at most  $3c^2$  fans are used to block some cell. Thus, we have at least  $3c^2$  interesting fans produced in the construction. The total number of vertices in the interesting fans is at least  $3c^2 \cdot 3c(1 + \lceil \ln 3c \rceil)$ . Recall that the total weight of vertices put into the table during the construction upper bounds the total size of all the interesting fans. We conclude that the weight of all the vertices put into the table during the construction is at least  $9c^3(1 + \lceil \ln 3c \rceil)$ . Observe that for each row  $i$  of the table and for each  $t = 1, \dots, 3c$  the number of vertices with weight greater or equal to  $\frac{3c}{t}$  is at most  $3c \cdot t$ . Indeed, let  $v$  be the first vertex put into the  $i$ -th row with weight at least  $\frac{3c}{t}$ . This means that there are at most  $t$  available cells in the  $i$ -th row at the time when  $v$  is put into the table. Since in each cell there are at most  $3c$  vertices we know that at most  $3c \cdot t$  more vertices may end up in the  $i$ -th row. Thus, there are at most  $3c$  vertices of weight  $\frac{3c}{1}$  and at most  $6c$  vertices of weight at least  $\frac{3c}{2}$  and so on, and it is easy to see that the total weight of the vertices in any row is at most

$$\sum_{t=1}^{3c} \frac{3c}{t} \cdot 3c = 9c^2 \sum_{t=1}^{3c} \frac{1}{t} < 9c^2(1 + \ln 3c).$$

If the game does not end before phase  $3c$  then vertices of total weight at least  $9c^3(1 + \lceil \ln 3c \rceil)$  are put into the table. However, total weight of vertices in any row is strictly smaller than  $9c^2(1 + \ln 3c)$ . Since there are  $c$  rows, we get a contradiction.

During the game Presenter introduced at most  $27c^3(1 + \lceil \ln 3c \rceil)$  vertices and forced Algorithm to use  $c$  colors. Inverting the function, we get that Presenter forces Algorithm to use  $\Omega\left(\frac{n}{\log n}^{\frac{1}{3}}\right)$  colors on graphs of size  $n$ .

To finish the proof, we need to argue that the constructed graph contains neither  $C_3$  nor  $C_5$  as a subgraph. For  $C_3$  observe that any vertex  $v$  introduced by Presenter has all the neighbors contained within one column (at the moment of introduction). Moreover, the vertex  $v$  itself ends up in the column to the right of

the column of its neighbors or is discarded. This implies that vertices in a single column form an independent set. Since the neighborhood of each vertex at the moment of introduction is an independent set the whole graph is triangle-free.

Now, assume to the contrary that  $C_5$  is contained in the constructed graph. As the graph does not contain  $C_3$ , the copy of  $C_5$  is an induced subgraph on some vertices  $v_0, v_1, v_2, v_3, v_4$ . We can assume, that all these vertices are put into the table. If it is otherwise, then we can exchange a discarded vertex  $u$  to any of the non discarded vertices, say  $w$ , in the same fan. All neighbors of  $u$  are also neighbors of  $w$  and we still get a  $C_5$ . We can choose  $v_2$  to be the vertex in the left most column of the five possibilities. By the construction, we know that  $v_1$  and  $v_3$  are introduced in a single fan of vertices adjacent to the group of vertices containing  $v_2$ . This implies that  $v_1$  and  $v_3$  are in the same column. All the neighbors of  $v_1$  in the columns to the left of  $v_1$  are in the same group (and column) as  $v_2$ . The same holds for  $v_3$ . If  $v_0$  and  $v_4$  were both in columns to the left of the column of  $v_1$  and  $v_3$  then they are in the same column as  $v_2$  and  $v_0$  is not adjacent to  $v_4$ . So at least one of  $v_0$  or  $v_4$  is in a column to the right of  $v_1$ . Say it is  $v_0$ . If  $v_4$  is in a column to the left of  $v_0$  then it is in the same column as both  $v_1$  and  $v_3$  and  $v_4$  is not adjacent to  $v_3$ . If  $v_4$  is in a column to the right of  $v_0$  then both  $v_0$  and  $v_3$  are in different columns to the left of  $v_4$ , and one of them is not adjacent to  $v_4$ .  $\square$

## References

1. Ajtai, M., Komlós, J., Szemerédi, E.: A note on Ramsey numbers. *Journal of Combinatorial Theory, Series A* **29**(3), 354–360 (1980)
2. Bianchi, M.P., Böckenhauer, H.-J., Hromkovič, J., Keller, L.: Online coloring of bipartite graphs with and without advice. *Algorithmica* **70**(1), 92–111 (2014)
3. Denley, T.: The independence number of graphs with large odd girth. *Electronic Journal of Combinatorics* **1**, R9, 12 p. (electronic) (1994)
4. Diwan, A.A., Kenkre, S., Vishwanathan, S.: Circumference, chromatic number and online coloring. *Combinatorica* **33**(3), 319–334 (2013)
5. Kierstead, H.A.: On-line coloring  $k$ -colorable graphs. *Israel Journal of Mathematics* **105**, 93–104 (1998)
6. Kierstead, H.A., Penrice, S.G., Trotter, W.T.: On-line coloring and recursive graph theory. *SIAM Journal on Discrete Mathematics* **7**(1), 72–89 (1994)
7. Kierstead, H.A., Penrice, S.G., Trotter, W.T.: On-line and first-fit coloring of graphs that do not induce  $P_5$ . *SIAM Journal on Discrete Mathematics* **8**(4), 485–498 (1995)
8. Kierstead, H.A., Trotter, W.T.: An extremal problem in recursive combinatorics. In: *Proceedings of the Twelfth Southeastern Conference on Combinatorics, Graph Theory and Computing*. *Congressus Numerantium*, vol. II, vol. 33, pp. 143–153 (1981)
9. Kim, J.H.: The Ramsey number  $R(3, t)$  has order of magnitude  $t^2 / \log t$ . *Random Structures and Algorithms* **7**(3), 173–207 (1995)
10. Krivelevich, M.: On the minimal number of edges in color-critical graphs. *Combinatorica* **17**(3), 401–426 (1997)
11. Lovász, L., Saks, M.E., Trotter, W.T.: An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics* **75**(1–3), 319–325 (1989)

# An On-line Competitive Algorithm for Coloring $P_8$ -free Bipartite Graphs

Piotr Micek<sup>1(✉)</sup> and Veit Wiechert<sup>2</sup>

<sup>1</sup> Theoretical Computer Science Department, Faculty of Mathematics  
and Computer Science, Jagiellonian University, Kraków, Poland  
`piotr.micek@tcs.uj.edu.pl`

<sup>2</sup> Technische Universität Berlin, Berlin, Germany  
`wiechert@math.tu-berlin.de`

**Abstract.** The existence of an on-line competitive algorithm for coloring bipartite graphs remains a tantalizing open problem. So far there are only partial positive results for bipartite graphs with certain small forbidden graphs as induced subgraphs, in particular for  $P_7$ -free bipartite graphs. We propose a new on-line competitive coloring algorithm for  $P_8$ -free bipartite graphs. Our proof technique improves the result, and shortens the proof, for  $P_7$ -free bipartite graphs.

## 1 Introduction

A *proper coloring* of a graph is an assignment of colors to its vertices such that adjacent vertices receive distinct colors. It is easy to devise an (linear time) algorithm 2-coloring bipartite graphs. Now, imagine that an algorithm receives vertices of a graph to be colored one by one knowing only the adjacency status of the vertex to vertices already colored. The color of a vertex must be fixed before the algorithm sees the next vertices and it cannot be changed afterwards. This kind of algorithm is called an *on-line* coloring algorithm.

Formally, an *on-line graph*  $(G, \pi)$  is a graph  $G$  with a permutation  $\pi$  of its vertices. An *on-line coloring algorithm*  $A$  takes an on-line graph  $(G, \pi)$ , say  $\pi = (v_1, \dots, v_n)$  as an input. It produces a proper coloring of the vertices of  $G$  where the color of a vertex  $v_i$ , for  $i = 1, \dots, n$ , depends only on the subgraph of  $G$  induced by  $v_1, \dots, v_i$ . It is convenient to imagine that consecutive vertices along  $\pi$  are revealed by some adaptive (malicious) adversary and the coloring process is a game between that adversary and an on-line algorithm.

Still, it is an easy exercise that if an adversary presents a bipartite graph and all the time the graph presented so far is connected then there is an on-line algorithm 2-coloring these graphs. But if an adversary can present a bipartite

---

V. Wiechert is supported by the Deutsche Forschungsgemeinschaft within the research training group ‘Methods for Discrete Structures’ (GRK 1408).

P. Micek is supported by Polish National Science Center UMO-2011/03/D/ST6/01370.

graph without any additional constraints then (s)he can trick out any on-line algorithm to use an arbitrary number of colors!

Indeed, there is a strategy for adversary forcing any on-line algorithm to use at least  $\lfloor \log n \rfloor + 1$  colors on a forest of size  $n$ . On the other hand, the First-Fit algorithm (that is an on-line algorithm coloring each incoming vertex with the least admissible natural number) uses at most  $\lfloor \log n \rfloor + 1$  colors on forests of size  $n$ . When the game is played on bipartite graphs, an adversary can easily trick out First-Fit and force  $\lceil \frac{n}{2} \rceil$  colors on a bipartite graph of size  $n$ . Lovász, Saks and Trotter [12] proposed a simple on-line algorithm (in fact as an exercise; see also [8]) using at most  $2 \log n + 1$  colors on bipartite graphs of size  $n$ . This is best possible up to an additive constant as Gutowski et al. [4] showed that there is a strategy for adversary forcing any on-line algorithm to use at least  $2 \log n - 10$  colors on a bipartite graph of size  $n$ .

For an on-line algorithm  $A$  by  $A(G, \pi)$  we mean the number of colors that  $A$  uses against an adversary presenting graph  $G$  with presentation order  $\pi$ .

An on-line coloring algorithm  $A$  is *competitive* on a class of graphs  $\mathcal{G}$  if there is a function  $f$  such that for every  $G \in \mathcal{G}$  and permutation  $\pi$  of vertices of  $G$  we have  $A(G, \pi) \leq f(\chi(G))$ . As we have discussed, there is no competitive coloring algorithm for forests. But there are reasonable classes of graphs admitting competitive algorithms, e.g., interval graphs can be colored on-line with at most  $3\chi - 2$  (where  $\chi$  is the chromatic number of the presented graph; see [11]) and comparability graphs can be colored on-line with a number of colors bounded by a tower function in terms of  $\chi$  (see [9]). Also classes of graphs defined in terms of forbidden induced subgraphs were investigated in this context. For example,  $P_4$ -free graphs (also known as cographs) are colored by First-Fit optimally, i.e. with  $\chi$  colors, since any maximal independent set meets all maximal cliques in a  $P_4$ -free graph. Also  $P_5$ -free graphs can be colored on-line with  $O(4^\chi)$  colors (see [10]). And to complete the picture there is no competitive algorithm for  $P_6$ -free graphs as Gyárfás and Lehel [6] showed a strategy for adversary forcing any on-line algorithm to use an arbitrary number of colors on bipartite  $P_6$ -free graphs.

Confronted with so many negative results, it is not surprising that Gyárfás, Király and Lehel [5] introduced a relaxed version of competitiveness of an on-line algorithm. The idea is to measure the efficiency of an on-line algorithm compared to the best on-line algorithm for a given input (instead of the chromatic number). Hence, the *on-line chromatic number* of a graph  $G$  is defined as

$$\chi_*(G) = \inf_A \max_\pi A(G, \pi),$$

where the infimum is taken over all on-line algorithms  $A$  and the maximum is taken over all permutation  $\pi$  of vertices of  $G$ . An on-line algorithm  $A$  is *on-line competitive* for a class of graphs  $\mathcal{G}$ , if there is a function  $f$  such that for every  $G \in \mathcal{G}$  and permutation  $\pi$  of vertices of  $G$  we have  $A(G, \pi) \leq f(\chi_*(G))$ .

Why are on-line competitive algorithms interesting? Imagine that you design an algorithm and the input graph is not known in advance. If your algorithm is on-line competitive then you have an insurance that whenever your algorithm

uses many colors on some graph  $G$  with presentation order  $\pi$  then any other on-line algorithm may be also forced to use many colors on the same graph  $G$  with some presentation order  $\pi'$  (and it includes also those on-line algorithms which are designed only for this single graph  $G$ !). The idea of comparing the outputs of two on-line algorithms directly (not via the optimal off-line result) is present in the literature. We refer the reader to [1], where a number of measures are discussed in the context of on-line bin packing problems. In particular, the relative worst case ratio, introduced there, is closely related to our setting for on-line colorings.

It may be true that there is an on-line competitive algorithm for all graphs. This is open as well for the class of all bipartite graphs. To the best of the authors knowledge, there is no promising approach for the negative answer for these questions. However, there are some partial positive results. Gyarfas and Lehel [7] have shown that First-Fit is on-line competitive for forests and it is even optimal in the sense that if First-Fit uses  $k$  colors on  $G$  then the on-line chromatic number of  $G$  is  $k$  as well. They also have shown [5] that First-Fit is competitive (with an exponential bounding function) for graphs of girth at least 5. Finally, Broersma, Capponi and Paulusma [2] proposed an on-line coloring algorithm for  $P_7$ -free bipartite graphs using at most  $8\chi_* + 8$  colors on graphs with on-line chromatic number  $\chi_*$ .

The contribution of this paper is the following theorem.

**Theorem 1.** *There is an on-line competitive algorithm that properly colors  $P_8$ -free bipartite graphs. Moreover, there are on-line coloring algorithms for bipartite graphs using at most*

- (i)  $4\chi_* - 2$  colors on  $P_7$ -free graphs,
- (ii)  $3(\chi_* + 1)^2$  colors on  $P_8$ -free graphs,

where  $\chi_*$  is the on-line chromatic number of the presented graph.

We wish to point out that we can improve the given bounds on the absolute values. But since this would need a more involved analysis and on the other hand the improvement would be small, we decided to present the weaker results. Furthermore, we can use our techniques to show that there is an on-line competitive algorithm for coloring  $P_9$ -free bipartite graphs. This result is not presented in this paper due to the page limitation.

## 2 Forcing Structure

In this section we introduce a family of bipartite graphs without long induced paths ( $P_6$ -free) and with arbitrarily large on-line chromatic number. All the on-line algorithms we are going to study have the property that whenever they use many colors on a graph  $G$  then  $G$  has a *large* graph from our family as an induced subgraph and therefore  $G$  has a large on-line chromatic number, as desired.

A connected bipartite graph  $G$  has a unique partition of vertices into two independent sets. We call these partition sets the *sides* of  $G$ . A vertex  $v$  in a

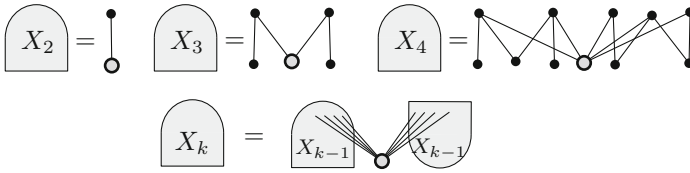


Fig. 1. Family of bipartite graphs

bipartite graph  $G$  is *universal* to a subgraph  $C$  of  $G$  if  $v$  is adjacent to all vertices of  $C$  in one of the sides of  $G$ .

Consider a family of connected bipartite graphs  $\{X_k\}_{k \geq 1}$  defined recursively as follows. Each  $X_k$  has a distinguished vertex called the *root*. The side of  $X_k$  containing the root of  $X_k$ , we call the *root side* of  $X_k$ , while the other side we call the *non-root side*.  $X_1$  is a single vertex being the root.  $X_2$  is a single edge with one of its vertices being the root.  $X_k$ , for  $k \geq 3$ , is a graph formed by two disjoint copies of  $X_{k-1}$ , say  $X_{k-1}^1$  and  $X_{k-1}^2$ , with no edge between the copies, and one extra vertex  $v$  adjacent to all vertices on the root side of  $X_{k-1}^1$  and all vertices on the non-root side of  $X_{k-1}^2$ . The vertex  $v$  is the root of  $X_k$ . Note that for each  $k$ , the root of  $X_k$  is adjacent to the whole non-root side of  $X_k$ , i.e., the root of  $X_k$  is universal in  $X_k$ . See Figure 1 for a schematic drawing of the definition of  $X_k$ .

A family of  $P_6$ -free bipartite graphs with arbitrarily large on-line chromatic number was first presented in [6]. The family  $\{X_k\}_{k \geq 1}$  was already studied in [3], in particular Claim 2 is proved there. Due to page limitation we omit the proof here. We encourage the reader to verify that  $X_k$  is  $P_6$ -free for  $k \geq 1$ .

**Claim 2.** *If  $G$  contains  $X_k$  as an induced subgraph, then  $\chi_*(G) \geq k$ .*

### 3 $P_7$ -Free Bipartite Graphs

In this section we present an on-line algorithm using at most  $4\chi_* - 2$  colors on  $P_7$ -free bipartite graphs with on-line chromatic number  $\chi_*$ . The algorithm itself, see Algorithm 1, is taken from [2,3] where it is called *BicolorMax* and proved that it uses at most  $2\chi_* - 1$  colors on  $P_6$ -free bipartite graphs and at most  $8\chi_* + 8$  colors on  $P_7$ -free bipartite graphs with on-line chromatic number  $\chi_*$ . Thus, we improve the bounding function for the  $P_7$ -free case and yet we present a much simpler proof.

Algorithm 1 uses two disjoint palettes of colors,  $\{a_n\}_{n \geq 1}$  and  $\{b_n\}_{n \geq 1}$ . In the following whenever the algorithm fixes a color of a vertex  $v$  we are going to refer to it by  $\text{color}(v)$ . Also for any set of vertices  $X$  we denote  $\text{color}(X) = \{\text{color}(x) \mid x \in X\}$ . We say that  $v$  has *color index*  $i$  if  $\text{color}(v) \in \{a_i, b_i\}$ .

Suppose an adversary presents a new vertex  $v$ . Let  $G_i[v]$  be the subgraph on the vertices presented so far that have a color from  $\{a_1, \dots, a_i, b_1, \dots, b_i\}$  and with one extra vertex, namely  $v$ , which is uncolored yet. Now,  $C_i[v]$  denotes the

connected component of  $G_i[v]$  containing vertex  $v$ . Furthermore, let  $C_i(v)$  be the graph  $C_i[v]$  without vertex  $v$ . The graph  $C_i(v)$  is not necessarily connected and in fact, as we will show, whenever  $v$  has color index  $k \geq 2$ ,  $C_i(v)$  contains at least 2 connected components for  $1 \leq i < k$ . Note that by these definitions it already follows that if  $w \in C_i(v)$  then  $C_j[w] \subseteq C_i(v)$  for all  $j \leq i$  since  $w$  is presented before  $v$  and has a smaller color index than  $v$ . We say that a color  $c$  is *mixed* in a connected induced subgraph  $C$  of  $G$  if  $c$  is used on vertices on both sides of  $C$ .

Now we are ready for a description of Algorithm 1.

---

**Algorithm 1.** On-line competitive for  $P_7$ -free bipartite graphs

---

```

an adversary introduces a new vertex  $v$ 
 $m \leftarrow \max \{i \geq 1 \mid a_i \text{ is mixed in } C_i[v]\} + 1$  //  $\max \{ \} := 0$ 
let  $I_1, I_2$  be the sides of  $C_m[v]$  such that  $v \in I_1$ 
if  $a_m \in \text{color}(I_2)$  then  $\text{color}(v) = b_m$ 
else  $\text{color}(v) = a_m$ 
    
```

---

It is not hard to see that Algorithm 1 colors bipartite graphs properly. We leave it as an exercise for the reader.

**Claim 3.** *Algorithm 1 gives a proper coloring of on-line bipartite graphs.*

The following claim is already proven in [2,3], for the sake of completeness we added it here. It is quite essential for the other proofs.

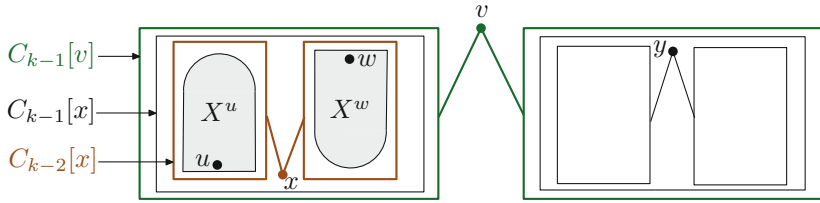
**Claim 4.** *Suppose an adversary presents a bipartite graph  $G$  to Algorithm 1. Let  $v \in G$  and let  $x, y$  be two vertices from opposite sides of  $C_i[v]$  both colored with  $a_i$ . Then  $x$  and  $y$  lie in different connected components of  $C_i(v)$ .*

*Proof.* Let  $v, x$  and  $y$  be like in the statement of the claim. We are going to prove that at any moment after the introduction of  $x$  and  $y$ ,  $x$  and  $y$  lie in different connected components of the subgraph spanned by vertices colored with  $a_1, b_1, \dots, a_i, b_i$ .

Say  $x$  is presented before  $y$ . First note that  $x \notin C_i[y]$  as otherwise  $x$  had to be on the opposite side to  $y$  (because it is on the opposite side at the time  $v$  is presented) and therefore  $y$  would receive color  $b_i$ . Now consider any vertex  $w$  presented after  $y$  and suppose the statement is true before  $w$  is introduced. If  $x \notin C_i[w]$  or  $y \notin C_i[w]$  then whatever color is used for  $w$  this vertex does not merge the components of  $x$  and  $y$  in the subgraph spanned by vertices presented so far and colored with  $a_1, b_1, \dots, a_i, b_i$ . Otherwise  $x, y \in C_i[w]$ . This means that color  $a_i$  is mixed in  $C_i[w]$  and therefore  $w$  receives a color with an index at least  $i + 1$ . Thus, the subgraph spanned by the vertices of the colors  $a_1, b_1, \dots, a_i, b_i$  stays the same and  $x$  and  $y$  remain in different connected components of this graph.

Since all vertices in  $C_i(v)$  are colored with  $a_1, b_1, \dots, a_i, b_i$ , we conclude that  $x$  and  $y$  lie in different components of  $C_i(v)$ . □





**Fig. 2.** Schematic drawing of some defined components. The color classes of  $G$  belong to the top and bottom side of the boxes. Vertex  $v$  has color index  $k$ . Vertices  $x$  and  $y$  certify that color  $a_{k-1}$  is mixed in  $C_{k-1}[v]$ . The component  $C_{k-1}[v]$  consists of both green boxes and vertex  $v$ , which is merging these.

As a consequence of Claim 4 it holds that if  $v$  has color index  $k \geq 2$  then  $C_i(v)$  is disconnected for all  $1 \leq i < k$ . This is simply because there are vertices  $x$  and  $y$  certifying that color  $a_i$  is mixed in  $C_i[v]$ . It also means that if we forget about the vertices presented so far that are not in  $C_i[v]$ , vertex  $v$  is merging independent connected subgraphs of  $G$  which are intuitively large in the case  $i = k - 1$  as they contain vertices ( $x$  and  $y$ ) with a high color index. See Figure 2 for a better understanding.

**Claim 5.** *Suppose an adversary presents a  $P_7$ -free bipartite graph to Algorithm 1. Let  $v$  be a vertex with color index  $k$  and  $1 \leq i < k$ . Then  $v$  is universal to all but possibly one component of  $C_i(v)$ .*

*Proof.* Suppose to the contrary that there are 2 components  $C_1$  and  $C_2$  in  $C_i(v)$  to which  $v$  is not universal. Then there are vertices  $v_1 \in C_1$  and  $v_2 \in C_2$  that both have distance at least 3 from  $v$  in  $C_i[v]$ . It follows that a shortest path connecting  $v_1$  and  $v$  in  $C_i[v]$ , combined with a shortest path connecting  $v$  and  $v_2$  in  $C_i[v]$  results in an induced path of length at least 7, a contradiction.  $\square$

**Theorem 6.** *Algorithm 1 uses at most  $4\chi_* - 2$  colors on  $P_7$ -free bipartite graphs with an on-line chromatic number  $\chi_*$ .*

*Proof.* Let  $G$  be a  $P_7$ -free bipartite graph that is presented by an adversary to Algorithm 1. Suppose color  $a_{2k}$  for some  $k \geq 1$  is used on a vertex  $v$  of  $G$ . We prove by induction on  $k$  that  $C_{2k-1}[v]$  contains  $X_{k+1}$  as an induced subgraph such that  $v$  corresponds to the root of  $X_{k+1}$ .

If  $k = 1$  then  $v$  is colored with  $a_2$  and  $v$  must have neighbors in  $C_1[v]$ . We embed  $X_2$  being a single edge onto  $v$  and its neighbor. So suppose  $k \geq 2$  and  $v$  is colored with  $a_{2k}$ . Since color  $a_{2k-1}$  is mixed in  $C_{2k-1}[v]$  there are vertices  $x$  and  $y$  of color  $a_{2k-1}$  lying on opposite sides of  $C_{2k-1}[v]$ . By Claim 4 vertices  $x$  and  $y$  lie in different components of  $C_{2k-1}(v)$ , say  $C^x$  and  $C^y$ . If we forget about the color index of  $v$  in Figure 2 then  $C^x$  corresponds to the left and  $C^y$  to the right green box. Using Claim 5 we conclude that  $v$  must be universal to at least one of  $C^x$  and  $C^y$ . We can assume that this is true for  $C^x$ . Since the color index of  $x$  is smaller than of  $v$ , we have that  $C_{2k-2}[x] \subseteq C^x$ . Now consider

vertices  $u$  and  $w$  in  $C_{2k-2}[x]$  certifying that color  $a_{2k-2}$  is mixed in  $C_{2k-2}[x]$ . Let  $C^u$  and  $C^w$  be the components of  $C_{2k-2}(x)$  containing  $u$  and  $v$ , respectively, which are distinct by Claim 4 (see left and right brown box in Figure 2). Observe that  $C_{2k-3}[u]$  and  $C_{2k-3}[w]$  are subgraphs of  $C^u$  and  $C^w$ , respectively. By the induction hypothesis there are induced copies  $X^u$  and  $X^w$  of  $X_k$  in  $C_{2k-3}[u]$  and  $C_{2k-3}[w]$ , respectively, such the roots correspond to  $u$  and  $v$  (see Figure 2). Since  $X^u \subseteq C^u$  and  $X^w \subseteq C^w$ , there is no edge between the copies and  $v$  is universal to both them. Using the fact that  $u$  and  $w$  appear on opposite sides of  $C_{2k-1}[v]$  we conclude that  $v$  together with  $X^u$  and  $X^w$  form an  $X_{k+1}$  with  $v$  being the root of it. This completes the induction.

Let now  $k \geq 1$  be maximal such that Algorithm 1 used the color  $a_{2k}$  on a vertex of  $G$ . It might be that also the colors  $b_{2k}, a_{2k+1}$  and  $b_{2k+1}$  are used, but not any  $a_j$  or  $b_j$  for  $j \geq 2k + 2$ . Thus, Algorithm 1 used at most  $4k + 2$  colors. On the other hand,  $G$  contains an  $X_{k+1}$  and by Claim 2 it follows that  $\chi_*(G) \geq k + 1$ . We conclude that Algorithm 1 used at most  $4k + 2 \leq 4\chi_*(G) - 2$  colors. □

### 4 $P_8$ -Free Bipartite Graphs

Inspired by Algorithm 1 and an argument from the previous section we present a new on-line algorithm for bipartite graphs and refer to it by Algorithm 2. In this section we prove that this algorithm is on-line competitive for  $P_8$ -free bipartite graphs.

Algorithm 2 uses three disjoint pallettes of colors,  $\{a_n\}_{n \geq 1}$ ,  $\{b_n\}_{n \geq 1}$  and  $\{c_n\}_{n \geq 1}$ . Similar to the case of  $P_7$ -freeness we make the following definitions. Whenever the algorithm fixes a color of a vertex  $v$  we are going to refer to it by  $\text{color}(v)$ . Also for any set of vertices  $X$  we denote  $\text{color}(X) = \{\text{color}(x) \mid x \in X\}$ . We say that a vertex  $v$  has *color index*  $i$ , if  $\text{color}(v) \in \{a_i, b_i, c_i\}$ .

Now, suppose an adversary presents a new vertex  $v$  of a bipartite graph  $G$ . Then let  $G_i[v]$  be the subgraph spanned by the vertices presented so far and colored with a color from  $\{a_1, \dots, a_i, b_1, \dots, b_i, c_1, \dots, c_i\}$  and vertex  $v$ , which is uncolored yet. With  $C_i[v]$  we denote the connected component of  $G_i[v]$  containing  $v$ . For convenience put  $C_0[v] = \{v\}$ . Furthermore, let  $C_i(v)$  be the graph  $C_i[v]$  without vertex  $v$ . For a vertex  $x$  in  $C_i(v)$  it will be convenient to denote by  $C_i^x(v)$  the connected component of  $C_i(v)$  that contains  $x$ . We say that a color  $c$  is *mixed* in a connected subgraph  $C$  of  $G$  if  $c$  is used on vertices on both sides of  $C$ .

Again, a proof for the proper coloring we leave as a fair exercise.

**Claim 7.** *Algorithm 2 gives a proper coloring of on-line bipartite graphs.*

To prove the following claim, it is enough to follow the lines of the proof for Claim 4.

**Claim 8.** *Suppose an adversary presents a bipartite graph  $G$  to Algorithm 2. Let  $v \in G$  and let  $x, y$  be two vertices from opposite sides of  $C_i[v]$  both colored with  $a_i$ . Then  $x$  and  $y$  lie in different connected components of  $C_i(v)$ .*

---

**Algorithm 2.** On-line competitive for  $P_8$ -free bipartite graphs

---

```

an adversary introduces a new vertex  $v$ 
 $m \leftarrow \max \{i \geq 1 \mid a_i \text{ is mixed in } C_i[v]\} + 1$  //  $\max \{ \} := 0$ 
let  $I_1, I_2$  be the sides of  $C_m[v]$  such that  $v \in I_1$ 
if  $a_m \in \text{color}(I_2)$  then  $\text{color}(v) = b_m$ 
else if  $c_m \in \text{color}(I_2)$  then  $\text{color}(v) = a_m$ 
else if  $\exists u \in I_1 \cup I_2$  and  $\exists u' \in I_2$  such that  $u$  has color index
 $j \geq m - \lfloor \sqrt{m-1} \rfloor$  and  $u'$  is universal to  $C_{j-1}[u]$  then  $\text{color}(v) = c_m$ 
else  $\text{color}(v) = a_m$ 

```

---

Now, whenever a vertex  $v$  has color index  $k \geq 2$ , let  $v_1$  and  $v_2$  be the first introduced vertices in  $C_{k-1}(v)$  that certify that color  $a_{k-1}$  is mixed in  $C_{k-1}(v)$ . By Claim 8 it follows that  $C_{k-1}^{v_1}(v)$  and  $C_{k-1}^{v_2}(v)$  are distinct (and in particular no edge is between them). In the following we will refer to  $v_1$  and  $v_2$  as the *children* of  $v$ .

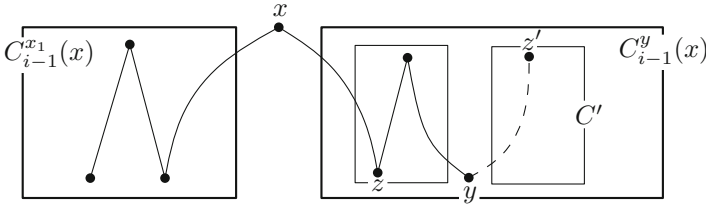
In the contrast to the  $P_7$ -free case, observe that  $v$  does not have to be universal to at least one of the components  $C_{k-1}^{v_1}(v)$  and  $C_{k-1}^{v_2}(v)$  if  $G$  is only  $P_8$ -free. However, as the next claim shows, we can expect a vertex on the other side of  $v$  in  $C_{k-1}(v)$  that is universal to some component in this case. This component will be only slightly smaller than  $C_{k-1}(v)$  whenever we use Claim 9 in our final proof. Also note that Algorithm 2 is inspired by this observation.

**Claim 9.** *Suppose an adversary presents a  $P_8$ -free bipartite graph  $G$  to Algorithm 2. Let  $x$  be a vertex with color index  $i \geq 2$ . Suppose that vertex  $y \in C_{i-1}(x)$ , with color index  $j$ , lies on the other side of  $x$  in  $G$  and  $y$  is not adjacent to  $x$ . Then one of the following holds:*

- (i)  $x$  has a child  $x'$  such that  $x$  is universal to  $C_{i-2}[x']$ , or
- (ii)  $x$  has a neighbor in  $C_{i-1}^y(x)$  that is universal to  $C_{j-1}[y]$ .

*Proof.* We can assume that  $y$  has color index  $j \geq 2$ , as otherwise  $C_{j-1}[y] = C_0[y] = \{y\}$  and vacuously any neighbor of  $x$  is universal to  $C_{j-1}[y]$  (as the side it must be adjacent to is empty). Let  $x_1$  and  $x_2$  be the children of  $x$ . By Claim 8 the components  $C_{i-1}^{x_1}(x)$  and  $C_{i-1}^{x_2}(x)$  are distinct. Vertex  $y$  is contained in at most one of them, say  $y \notin C_{i-1}^{x_1}(x)$ , and therefore  $C_{i-1}^{x_1}(x)$  and  $C_{i-1}^y(x)$  are distinct. In order to prove the claim suppose that (i) is not satisfied. Then  $x$  is not universal to  $C_{i-2}[x_1]$  and in particular not to  $C_{i-1}^{x_1}(x)$ . It follows that there is an induced path of length 4 ending at  $x$  in  $C_{i-1}^{x_1}[x]$ . As  $G$  is  $P_8$ -free we conclude that  $C_{i-1}^y[x]$  does not contain an induced path of length 5 with one endpoint in  $x$ . With this observation in mind we will prove (ii) now.

First, let us consider the case that  $x$  has a neighbor  $z$  in  $C_{j-1}[y] \subseteq C_{i-1}^y(x)$  (see Figure 3 for this case). Since  $x$  and  $y$  are not adjacent we have  $y \neq z$ . As  $C_{j-1}^z[y]$  is connected, there is an induced path  $P$  connecting  $x$  and  $y$  that has only vertices of  $C_{j-1}^z(y)$  as inner vertices. Clearly,  $P$  has even length at least 4. Now the color index of  $y$ , namely  $j \geq 2$ , assures the existence of a mixed pair in  $C_{j-1}[y]$  and with Claim 8 it follows that  $C_{j-1}(y)$  has at least two



**Fig. 3.** Claim 9: Situation in which  $x$  has a neighbor  $z$  in  $C_{j-1}[y]$

connected components. In particular, there is a component  $C'$  of  $C_{j-1}(y)$  other than  $C_{j-1}^z(y)$ . Clearly,  $y$  has a neighbor  $z'$  in  $C'$ , which we use to prolong  $P$  at  $y$ . Since there is no edge between  $C_{j-1}^z(y)$  and  $C'$ , vertex  $z'$  is not adjacent to the inner vertices of  $P$ . And as  $G$  is bipartite  $z'$  cannot be adjacent to  $x$ . We conclude the existence of an induced path of length 5 in  $C_{i-1}^y[x]$  with  $x$  and  $z'$  being its endpoints, a contradiction.

Second, we consider the case that  $x$  has no neighbor in  $C_{j-1}[y]$ . By our assumptions a shortest path connecting  $x$  and  $y$  in  $C_{i-1}^y[x]$  must have length exactly 4. Let  $P = (x, r, s, y)$  be such a path. We claim that vertex  $r$  is universal to  $C_{j-1}[y]$ . Suppose to contrary that there is a vertex  $s'$  in  $C_{j-1}[y]$  which is on the other side of  $y$  and which is not adjacent to  $r$ . Let  $Q = (y, s_1, r_1, \dots, s_{\ell-1}, r_{\ell-1}, s_{\ell} = s')$  be a shortest path connecting  $y$  and  $s'$  in  $C_{j-1}[y]$ . For convenience put  $s_0 = s$ . Now we choose the minimal  $m \geq 0$  such that  $r$  is adjacent to  $s_m$  but not to  $s_{m+1}$ . Such an  $m$  exists since  $r$  is adjacent to  $s_0 = s$  but not to  $s_{\ell}$ . If  $m = 0$  then the path  $(x, r, s, y, s_1)$  is an induced path of length 5 and if  $m > 0$  then the path  $(x, r, s_m, r_m, s_{m+1})$  has length 5 and is induced unless  $x$  and  $r_m$  are adjacent. But the latter is not possible since  $x$  has no neighbor in  $C_{j-1}[y]$ . Thus, in both cases we get a contradiction and we conclude that  $r$  is universal to  $C_{j-1}[y]$ .  $\square$

In the following we write  $v \rightarrow_i w$  for  $v, w \in G$ , if there is a sequence  $v = x_1, \dots, x_j = w$  with  $j \leq i$  and  $x_{\ell+1}$  is a child of  $x_{\ell}$ , for all  $\ell \in \{1, \dots, j-1\}$ . Moreover, we define  $S_i(v) = \{w \mid v \rightarrow_i w\}$ .

We make some immediate observations concerning this definition. Let  $v \in G$  be a vertex with color index  $k \geq 2$ . Then all vertices in  $S_i(v)$  have color index at least  $k - i + 1$ . Furthermore, each vertex in  $S_i(v)$  is connected to  $v$  by a path in  $G$  and all vertices in the path, except  $v$ , have color index at most  $k - 1$ . This proves that  $S_i(v) \subseteq C_{k-1}[v]$ , for all  $i \geq 1$ . Note also that if  $v_1$  and  $v_2$  are the children of  $v$  then we have  $S_i(v) = \{v\} \cup S_{i-1}(v_1) \cup S_{i-1}(v_2)$  and  $S_{i-1}(v_1) \subseteq C_{k-1}^{v_1}(v)$ ,  $S_{i-1}(v_2) \subseteq C_{k-1}^{v_2}(v)$ . By Claim 8, we get that  $C_{k-1}^{v_1}(v)$  and  $C_{k-1}^{v_2}(v)$  are distinct. In particular,  $S_{i-1}(v_1)$  and  $S_{i-1}(v_2)$  are disjoint and there is no edge between them.

For a vertex  $v \in G$ ,  $S_i(v)$  is *complete* in  $G$  if for every  $u, w \in S_i(v)$  such that  $u \rightarrow_i w$  and  $u, w$  lying on opposite sides of  $G$ , we have  $u$  and  $w$  being adjacent in  $G$ . Note that  $v$  is a universal vertex in  $S_i(v)$ , provided  $S_i(v)$  is complete.

**Claim 10.** *Suppose an adversary presents a bipartite graph  $G$  to Algorithm 2. Let  $v \in G$  be a vertex with color index  $k$  and let  $k \geq i \geq 1$ . If  $S_i(v)$  is complete then  $S_i(v)$  contains an induced copy of  $X_i$  in  $G$  with  $v$  being the root of the copy.*

*Proof.* We prove the claim by induction on  $i$ . For  $i = 1$  we work with  $S_1(v)$  and  $X_1$  being graphs with one vertex only, so the statement is trivial. For  $i \geq 2$ , let  $v_1$  and  $v_2$  be the children of  $v$ . Recall that  $S_i(v) = \{v\} \cup S_{i-1}(v_1) \cup S_{i-1}(v_2)$ . Since  $S_i(v)$  is complete it also follows that  $S_{i-1}(v_1)$  and  $S_{i-1}(v_2)$  are complete. So by the induction hypothesis there are induced disjoint copies  $X_{i-1}^1, X_{i-1}^2$  of  $X_{i-1}$  in  $S_{i-1}(v_1)$  and  $S_{i-1}(v_2)$ , respectively, and rooted in  $v_1, v_2$ , respectively. Recall that  $S_{i-1}(v_1)$  and  $S_{i-1}(v_2)$  are disjoint and there is no edge between them. Thus, the copies  $X_{i-1}^1$  and  $X_{i-1}^2$  of  $X_{i-1}$  are disjoint and there is no edge between them, as well. Since  $S_i(v)$  is complete  $v$  is universal to both of the copies, and since  $v_1$  and  $v_2$  lie on opposite sides in  $G$  we get that the vertices of  $X_{i-1}^1 \cup X_{i-1}^2 \cup \{v\}$  induce a copy of  $X_i$  in  $G$ .  $\square$

**Claim 11.** *Suppose an adversary presents a  $P_8$ -free bipartite graph  $G$  to Algorithm 2 and suppose vertex  $v$  is colored with  $a_k$ . Then  $C_k[v]$  contains an induced copy of  $X_{\lfloor \sqrt{k} \rfloor}$  such that its root lies on the same side as  $v$  in  $G$ .*

*Proof.* The claim is easy to verify for  $1 \leq k \leq 8$ . So suppose that  $k \geq 9$ . Consider the set  $S_{\lfloor \sqrt{k} \rfloor}(v)$ . If it is complete then by Claim 10 we get an induced copy of  $X_{\lfloor \sqrt{k} \rfloor}$  with a root mapped to  $v$ . So we are done.

From now on we assume that  $S_{\lfloor \sqrt{k} \rfloor}(v)$  is not complete. Let  $(I_1, I_2)$  be the bipartition of  $C_k[v]$  such that  $v \in I_1$ . First, we will prove that there are vertices  $z, z' \in C_k[v]$  such that  $z' \in I_1$ ,  $z$  has color index  $\ell \geq k - \lfloor \sqrt{k-1} \rfloor$  and  $z'$  is universal to  $C_{\ell-1}[z]$ . To do so we consider the reason why Algorithm 2 colors  $v$  with  $a_k$  instead of  $b_k$  or  $c_k$ .

The first possible reason is that the second if-condition of the algorithm is satisfied, that is, there is a vertex  $u \in I_2$  colored with  $c_k$ . Now  $u$  can only receive color  $c_k$  if there are vertices  $w, w' \in C_k[u]$  such that  $w'$  is on the other side of  $u$  in  $C_k[u]$ ,  $w$  has color index  $j \geq k - \lfloor \sqrt{k-1} \rfloor$  and  $w'$  is universal to  $C_{j-1}[w]$ . Since  $C_k(u) \subseteq C_k(v)$  and  $u \in I_2$  we have  $w' \in I_1$ . Therefore,  $w$  and  $w'$  prove the existence of vertices that we are looking for in this case.

The second reason for coloring  $v$  with  $a_k$  is that Algorithm 2 reaches its last line. In particular this means, that there is no vertex of color  $a_k$  or  $c_k$  in  $I_2$ . Now we are going to make use of the fact that  $S_{\lfloor \sqrt{k} \rfloor}(v)$  is not complete. There are vertices  $x, y \in S_{\lfloor \sqrt{k} \rfloor}(v) \subseteq C_k[v]$  such that  $x \rightarrow_{\lfloor \sqrt{k} \rfloor} y$ , vertices  $x$  and  $y$  lie on different sides of  $C_k[v]$  and are not adjacent. Let  $i$  and  $j$  be the color indices of  $x$  and  $y$ , respectively. Note that  $k \geq i > j \geq k - \lfloor \sqrt{k} \rfloor + 1$ . By Claim 9 vertex  $x$  has a child  $x'$  such that  $x$  is universal to  $C_{i-2}[x']$  or  $x$  has a neighbor  $r \in C_{i-1}^y(x)$  that is universal to  $C_{j-1}[y]$ . In the first case let  $w'$  be  $x$  and  $w$  be  $x'$ , and in the second case we set  $w'$  to be  $r$  and  $w$  to be  $y$ . Then, in both cases it holds that  $w'$  is universal to  $C_{\ell-1}[w]$ , where  $\ell$  is the color index of  $w$ . Furthermore we have

$$\ell \geq \min\{i-1, j\} \geq k - \lfloor \sqrt{k} \rfloor + 1 \geq k - \lfloor \sqrt{k-1} \rfloor$$

for all  $k \geq 1$ . Since  $w' \in C_k[v]$  we have  $w' \in I_1$  or  $w' \in I_2$ . However, the latter is not possible as otherwise  $w$  and  $w'$  would fulfill the conditions of the third if-statement in Algorithm 2, which contradicts the fact that Algorithm 2 reached the last line for  $v$ . We conclude that  $w' \in I_1$ , which completes the proof of our subclaim.

So suppose we have vertices  $z$  and  $z'$  like in our subclaim. Let  $z_1$  and  $z_2$  be the children of  $z$  (they exist as  $\ell \geq 2$ ). Both vertices received color  $a_{\ell-1}$  and are on different sides of  $G$ . By the induction hypothesis  $C_{\ell-1}[z_1]$  and  $C_{\ell-1}[z_2]$  contain a copy of  $X_{\lfloor \sqrt{\ell-1} \rfloor}$  such that the roots are on the same side as  $z_1$  and  $z_2$ , respectively. Since there is no edge between  $C_{\ell-1}[z_1]$  and  $C_{\ell-1}[z_2]$  and both are contained in  $C_\ell[z]$ , it follows that  $z'$  together with the copies of  $X_{\lfloor \sqrt{\ell-1} \rfloor}$  induce a copy of  $X_{\lfloor \sqrt{\ell-1} \rfloor + 1}$  that has  $z'$  as its root. Since  $C_\ell[z]$  is contained in  $C_k[v]$  and  $z'$  is on the same side as  $v$  and since

$$\lfloor \sqrt{\ell-1} \rfloor + 1 \geq \left\lfloor \sqrt{k - \lfloor \sqrt{k-1} \rfloor - 1} \right\rfloor + 1 \geq \lfloor \sqrt{k} \rfloor,$$

for all  $k \geq 1$ , this completes the proof. □

Now we are able to prove our main theorem.

**Theorem 12.** *On each on-line  $P_8$ -free bipartite graph  $(G, \pi)$ , Algorithm 2 uses at most  $3(\chi_*(G) + 1)^2$  colors.*

*Proof.* Let  $k$  be the highest color index on the vertices of  $G$ . By the definition of Algorithm 2 the color  $a_k$  appears on some vertex of  $G$ . Using Claim 11 it follows that  $G$  contains  $X_{\lfloor \sqrt{k} \rfloor}$ . Now, by Claim 2,  $\chi_*(G) \geq \lfloor \sqrt{k} \rfloor$  and therefore Algorithm 2 uses at most  $3k \leq 3(\lfloor \sqrt{k} \rfloor + 1)^2 \leq 3(\chi_*(G) + 1)^2$  colors. □

## References

1. Boyar, J., Favrholt, L.M.: The relative worst order ratio for online algorithms. *ACM Trans. Algorithms* **3**(2), 22–24 (2007)
2. Broersma, H.J., Capponi, A., Paulusma, D.: A new algorithm for on-line coloring bipartite graphs. *SIAM Journal on Discrete Mathematics* **22**(1), 72–91 (2008)
3. Broersma, H., Capponi, A., Paulusma, D.: On-line coloring of  $H$ -free bipartite graphs. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.) *CIAC 2006*. LNCS, vol. 3998, pp. 284–295. Springer, Heidelberg (2006)
4. Gutowski, G., Kozik, J., Micek, P.: Lower bounds for on-line graph colorings (submitted)
5. Gyárfás, A., Király, Z., Lehel, J.: On-line competitive coloring algorithms. Technical report TR-9703-1 (1997). <http://www.cs.elte.hu/tr97/tr9703-1.ps>
6. Gyárfás, A., Lehel, J.: On-line and first fit colorings of graphs. *Journal of Graph Theory* **12**(2), 217–227 (1988)
7. Gyárfás, A., Lehel, J.: First fit and on-line chromatic number of families of graphs. *Ars Combin.* **29**(C), 168–176 (1990); Twelfth British Combinatorial Conference, Norwich (1989)

8. Kierstead, H.A.: Recursive and on-line graph coloring. In: Handbook of Recursive Mathematics. Stud. Logic Found. Math., vol. 2, vol. 139, pp. 1233–1269. North-Holland, Amsterdam (1998)
9. Kierstead, H.A., Penrice, S.G., Trotter, W.T.: On-line coloring and recursive graph theory. *SIAM Journal on Discrete Mathematics* **7**(1), 72–89 (1994)
10. Kierstead, H.A., Penrice, S.G., Trotter, W.T.: On-line and first-fit coloring of graphs that do not induce  $P_5$ . *SIAM Journal on Discrete Mathematics* **8**(4), 485–498 (1995)
11. Kierstead, H.A., Trotter, W.T.: An extremal problem in recursive combinatorics. In: Proceedings of the Twelfth Southeastern Conference on Combinatorics, Graph Theory and Computing. *Congressus Numerantium*. vol. II, vol. 33, pp. 143–153 (1981)
12. Lovász, L., Saks, M.E., Trotter, W.T.: An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics* **75**(1–3), 319–325 (1989)

# Bounds on Double-Sided Myopic Algorithms for Unconstrained Non-monotone Submodular Maximization

Norman Huang and Allan Borodin<sup>(✉)</sup>

University of Toronto, Toronto, Canada  
{huangyum, bor}@cs.toronto.edu

**Abstract.** Unconstrained submodular maximization captures many NP-hard combinatorial optimization problems, including MAX-CUT, MAX-DICUT, and variants of facility location problems. Recently, Buchbinder *et al.* [4] presented a surprisingly simple linear time randomized *greedy-like* online algorithm that achieves a constant approximation ratio of  $\frac{1}{2}$ , matching optimally the hardness result of Feige *et al.* [10]. Motivated by the algorithm of Buchbinder *et al.*, we introduce a precise algorithmic model called *double-sided myopic algorithms*. We show that while the algorithm of Buchbinder *et al.* can be realized as a randomized online double-sided myopic algorithm, no such deterministic algorithm, even with adaptive ordering, can achieve the same approximation ratio. With respect to the MAX-DICUT problem, we relate the Buchbinder *et al.* algorithm and our myopic framework to the online algorithm and inapproximation of Bar-Noy and Lampis [2].

## 1 Introduction

Submodularity emerges in natural settings such as economics, algorithmic game theory, and operations research; many combinatorial optimization problems can be abstracted as the maximization/minimization of submodular functions. In particular, submodular maximization generalizes NP-hard problems such as MAX-(DI)CUT [1, 14, 15], MAX-COVERAGE [8], expected influence in a social network [17] and facility location problems [6, 7]. As such, a number of approximation heuristics have been studied in the literature. For monotone submodular functions, maximization under a cardinality constraint can be achieved greedily with an approximation ratio of  $(1 - \frac{1}{e})$  [20], which is in fact optimal in the *value oracle model* [19]. The same approximation ratio is obtainable for the more general matroid constraints [5, 12], as well as knapsack constraints [18, 24]. We limit our discussion to unconstrained non-monotone submodular maximization (USM) — typical examples of which include MAX-CUT and MAX-DICUT. We refer the reader to our paper version [16] for a more comprehensive discussion on related works.

Recently, linear time (linear in counting one step per oracle call) *double-sided greedy* algorithms were developed by Buchbinder *et al.* [4] for the general USM. The deterministic version of their algorithm achieves an approximation



ratio of  $\frac{1}{3}$ , while the randomized version achieves  $\frac{1}{2}$  in expectation - improving upon the  $\frac{2}{5}$  randomized local-search approach in [10], and the 0.42 simulated-annealing technique in [11, 13], in terms of approximation ratio, time complexity and arguably, algorithmic simplicity. While the hardness result of Feige *et al.* [10] implies optimality of the randomized algorithm, the gap between the deterministic and randomized variants remains an open problem. More specifically, is there any de-randomization that would preserve both the greedy aspect of the algorithm as well as the approximation? To address this question, we adapt the priority algorithms framework of Borodin *et al.* [3]. Specifically, we define a *double-sided myopic algorithms* framework that captures the Buchbinder *et al.* algorithms; and show that no deterministic algorithm in this framework can de-randomize the  $\frac{1}{2}$ -ratio double-sided greedy algorithm.

In addition to [4], Bar-Noy and Lampis [2] give a  $\frac{1}{3}$  deterministic online greedy algorithm for MAX-DICUT matching the deterministic approximation obtained by Buchbinder *et al.* for USM. They also give an improved  $\frac{2}{3\sqrt{3}}$  approximation for MAX-DICUT on DAGs, and a precise online model with respect to which this approximation is optimal. Feige and Jozeph [9] introduce *oblivious online algorithms* for MAX-DICUT and give a 0.483 approximation and a 0.4899 inapproximation for randomized oblivious algorithms. Independent of our work, Paul, Poloczek and Williamson [22] have very recently derived a number of deterministic algorithms, and deterministic and randomized inapproximations for MAX-DICUT with respect to the priority algorithm framework.

---

**Algorithm 1. DeterministicUSM( $f, \mathcal{N}$ )**

---

```

1:  $S_0 \leftarrow \emptyset, T_0 \leftarrow \mathcal{N}$ 
2: for  $i = 1$  to  $n$  do
3:    $a_i \leftarrow f(S_{i-1} \cup \{u_i\}) - f(S_{i-1})$ 
4:    $b_i \leftarrow f(T_{i-1} \setminus \{u_i\}) - f(T_{i-1})$ 
5:   if  $a_i \geq b_i$  then
6:      $S_i \leftarrow S_{i-1} \cup \{u_i\}, T_i \leftarrow T_{i-1}$ 
7:   else
8:      $T_i \leftarrow T_{i-1} \setminus \{u_i\}, S_i \leftarrow S_{i-1}$ 
9:   end if
10: end for
11: return  $S_n$ 

```

---

**1.1 Basic Definitions**

A set function  $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}$  is *submodular* if for any  $S, T \subseteq \mathcal{N}$ ,  $f(S \cup T) + f(S \cap T) \leq f(S) + f(T)$ . We say  $f$  is *monotone* if  $f(S) \leq f(T)$  for all  $S \subseteq T \subseteq \mathcal{N}$ , and *non-monotone* otherwise. In USM, the goal is to find a subset  $S \subseteq \mathcal{N} = \{u_1, \dots, u_n\}$  maximizing  $f(S)$  for a specified submodular function  $f$ . In general, since the specification of  $f$  requires knowing its value on all possible subsets,  $f$  is accessed via a value oracle which given  $X \subseteq \mathcal{N}$ , returns  $f(X)$ . We state the  $\frac{1}{3}$  deterministic double greedy algorithm by Buchbinder *et al.* in Algorithm 1.

## 1.2 Our Contribution

We introduce a formalization of *double-sided myopic algorithms* - an adaptation of the priority framework under a *restricted* value oracle model. To make this precise, we will introduce three types of *relevant oracle queries*. In Proposition 1 and 2 we show that the double-sided greedy algorithms of Buchbinder *et al.* can be realized as online double-sided myopic algorithms. Moreover, our framework also captures the online model of Bar-Noy and Lampis for MAX-DICUT, upon which Theorem 1 follows essentially by definition.

---

### Algorithm 2. OnlineMyopic( $f, \bar{f}$ )

---

```

1:  $X_0 \leftarrow \emptyset, Y_0 \leftarrow \emptyset$ 
2: for  $i = 1$  to  $n$  do
3:    $a_i \leftarrow f(X_{i-1} \cup \{u_i\}) - f(X_{i-1})$ 
4:    $b_i \leftarrow \bar{f}(Y_{i-1} \cup \{u_i\}) - \bar{f}(Y_{i-1})$ 
5:   if  $a_i \geq b_i$  then
6:      $X_i \leftarrow X_{i-1} \cup \{u_i\}, Y_i \leftarrow Y_{i-1}$ 
7:   else
8:      $Y_i \leftarrow Y_{i-1} \cup \{u_i\}, X_i \leftarrow X_{i-1}$ 
9:   end if
10: end for
11: return  $X_n$ 

```

---

**Proposition 1.** *The deterministic double greedy algorithm in [4] can be modeled by Algorithm 2, a Q-Type 1 online double-sided myopic algorithm.*

**Proposition 2.** *The randomized double greedy algorithm in [4] can be modeled by a random-choice Q-Type 1 online double-sided myopic algorithm.*

**Theorem 1.** *No deterministic online double-sided myopic algorithm (with respect to our strongest oracle model Q-Type 3) can achieve a competitive ratio of  $\frac{2}{3\sqrt{3}} + \epsilon \approx 0.385 + \epsilon$  for any  $\epsilon > 0$  for USM.*

As our main contribution, we extend to the class of (deterministic) fixed and adaptive priority algorithms in Theorem 2 and 3. We construct via linear programming submodular functions and corresponding adversarial strategies that would force an inapproximability ratio strictly less than  $\frac{1}{2}$ . In terms of oracle restrictiveness, our inapproximation holds for the *all subsets query* model (Q-Type 3) for the fixed case and the *already attained partial solution query* model (Q-Type 2), which is more powerful than what is sufficient to achieve the  $\frac{1}{2}$  ratio by the online randomized greedy algorithm (Q-Type 1). Due to space constraint we refer to [16] for missing proofs as well as further discussion.

**Theorem 2.** *There exists a problem instance such that no fixed priority double-sided myopic algorithm with respect to oracle model Q-Type 3 can achieve an approximation ratio better than 0.450.*

**Theorem 3.** *There exists a problem instance such that no adaptive priority double-sided myopic algorithm with respect to oracle model Q-Type 2 can achieve an approximation ratio better than 0.432.*

## 2 The Double-Sided Myopic Algorithms Framework

By integrating a *restricted* value oracle model in a priority framework, we propose a general class of *double-sided myopic algorithms* that captures the Buchbinder *et al.* double-sided greedy algorithm. Using a pair of complementary objective functions, we rephrase the double-sided procedure (*i.e.* simultaneously evolving a bottom-up and a top-down solution) as a single sided sweep common to most greedy algorithms; and this facilitates an adaptation of a priority-like framework. We must also specify how input items are represented and accessed. The generality of USM raises some representational issues, which we address by employing a *marginal value* representation that is compatible with both the value oracle model and the priority framework.

While a precise description of a data item is necessary in determining the input ordering and in quantifying the availability of information in the decision step, the value oracle model measures complexity in terms of information access. An apparent incompatibility arises when an exact description of a data item can trivialize query complexity - as in the case of the *marginal value* representation, where exponentially many queries are needed to fully describe an item when  $f$  is an arbitrary submodular function. For this reason, we propose a hierarchy of oracle restrictions that categorizes the concept of myopic *short-sightedness*, while preserving the fundamental characteristics of the priority framework.

### 2.1 Value Oracle and the Marginal Value Representation

Since  $f$  may not have a succinct encoding, to avoid having an exponential sized input we employ a *value oracle*. That is, given a query  $S \subseteq \mathcal{N}$ , the value oracle returns the value of  $f(S)$ . Abusing notation, we will interchangeably refer to  $f$  as the objective function (*i.e.* when referring to  $f(S)$  as a real number) and as the value oracle (*i.e.* when an algorithm submits a query to  $f$ ).

We also introduce for notational convenience a complementary oracle  $\bar{f}$ , such that  $\bar{f}(X) = f(\mathcal{N} \setminus X)$ . This allows us to express the double-sided myopic algorithm similar to the priority setting in [3], where the solution set  $X$  is constructed using only locally available information. In other words, the introduction of  $\bar{f}$  allows access to  $f(\mathcal{N} \setminus X)$  using  $X$  (which is composed of items that have already been considered) as query argument, instead of  $\mathcal{N} \setminus X$ . The **double-sidedness** of the framework follows in the sense that  $f$  and  $\bar{f}$  can be simultaneously accessed.

We wish to model *greedy-like* algorithms that process the problem instance item by item. But what is an item when considering an arbitrary submodular function? While the natural choice is that an item is an element of  $\mathcal{N}$  (to include or not include in the solution  $S$ ), the input of USM is the objective function itself, drawn from the family  $\mathcal{F} = \{f | f : 2^{\mathcal{N}} \rightarrow \mathbb{R}\}$  of all submodular set

functions over a fixed  $\mathcal{N}$ . To address this issue, we propose the *marginal value representation*, in which  $f$  is instilled into the elements of  $\mathcal{N}$ . Specifically, we describe a data item as an element  $u \in \mathcal{N}$ , plus a list of marginal differences  $\rho(u|S) = f(S \cup \{u\}) - f(S)$ , and  $\bar{\rho}(u|S) = \bar{f}(S \cup \{u\}) - \bar{f}(S)$  for every subset  $S \subseteq \mathcal{N}$ . A complete representation in this form is not only exponential in  $|\mathcal{N}|$ , it leads to a trivial optimal greedy algorithm. Therefore we assume that an oracle query must be made by the algorithm in order to access each marginal value. However, if certain natural constraints are imposed on allowable oracle queries during the computation, then the model allows us to justify when input items are indistinguishable. In fact, our inapproximation results will not be based on bounding of the number of oracle queries but rather by the restricted myopic nature of the algorithm.

### 2.2 Classes of Relevant Oracle Queries

The **myopic** condition of our framework is imposed both by the nature of the ordering in the priority model, which we describe later on, as well as by restricting the algorithm to make only *relevant* oracle queries. To avoid ambiguity, assume the oracles are given in terms of  $f$  and  $\bar{f}$ , with  $\rho$  and  $\bar{\rho}$  being used purely for notational simplicity. That is, when we say that the algorithm queries  $\rho(u|S)$ , we assume that it queries both  $f(S \cup \{u\})$  and  $f(S)$ . At iteration  $i$ , define (respectively)  $X_{i-1}$  and  $Y_{i-1}$  to be the currently accepted and rejected sets, and  $u_i$  to be the next item considered by the algorithm. We introduce three models of relevant oracle queries in order of increasing information.<sup>1</sup>

Next attainable partial solution query (Q-Type 1)

The algorithm is permitted to only query  $f(X_{i-1} \cup \{u_i\})$  and  $\bar{f}(Y_{i-1} \cup \{u_i\})$ , *i.e.* the values of the next possible partial solution. In all models, the algorithm can then use this information in any way. For example, the deterministic algorithm of Buchbinder *et al.* greedily chooses to add  $u_i$  to  $X_{i-1}$  if  $\rho(u_i|X_{i-1}) = a \geq b = \bar{\rho}(u_i|Y_{i-1})$ . In our model, the decision about  $u_i$  can be any (even non-computable) function of  $a$  and  $b$  and the “history” of the algorithm thus far.

Already attained partial solutions query (Q-Type 2)

In this model we allow queries of the form  $\rho(u_i|X_j)$  and  $\bar{\rho}(u_i|Y_j)$  for all  $j < i$ .

All subsets query (Q-Type 3)

Here, the algorithm can query  $\rho(u_i|S)$  and  $\bar{\rho}(u_i|S)$  for every  $S \subseteq X_{i-1} \cup Y_{i-1}$ . Note that in this very general model, the algorithm can potentially query exponentially many sets so that in principle such algorithms are not subject to the  $\frac{1}{2}$  inapproximation result of Feige *et al* [9].

### 2.3 Internal Memory or History

We define an algorithm’s *internal memory* or *history* as a record of the following:

<sup>1</sup> See Section 5 for further motivation.

- All previously considered items and the decisions made for these items.
- The outcomes of all previous relevant query results.
- Anything deducible from previously considered items, decisions and relevant queries. Consequently, the algorithm can rule out all submodular functions contradicting the observed marginal values for previously considered items.

At the start of iteration  $i$ , allow the algorithm to perform any possible relevant queries. Let  $\mathcal{N}_{i-1} = X_{i-1} \cup Y_{i-1}$  be the set of all previously seen (and decided upon) items. The algorithm’s internal memory under each query restriction type contains the following:

Q-Type 1 myopic model

- The decision made for every  $u \in \mathcal{N}_{i-1}$ .
- $\rho(u_j|X_{j-1})$  and  $\bar{\rho}(u_j|Y_{j-1})$  for  $0 < j \leq i$ .

Q-Type 2 myopic model

- The decision made for every  $u \in \mathcal{N}_{i-1}$ .
- $\rho(u_j|X_k)$  and  $\bar{\rho}(u_j|Y_k)$  for  $0 \leq k < j \leq i$ .

Q-Type 3 myopic model

- The decision made for every  $u \in \mathcal{N}_{i-1}$ .
- $\rho(u|S)$  and  $\bar{\rho}(u|S)$  for all  $u \in \mathcal{N}_i$  and all  $S \subseteq \mathcal{N}_i$ .

## 2.4 Priority Models

Finally, we specify the order in which input items are considered. We categorize double-sided myopic algorithms into the following subclasses: **online**, **fixed** priority, and **adaptive** priority. For all templates, the algorithm makes an irrevocable decision for the current item based on the history of relevant queries. Initially,  $|\mathcal{N}|$  is the only accessible information. Revealing the input length allows for a broader and potentially more powerful class of algorithms compared to the priority framework. That is, an adversary cannot abruptly end a computation. We begin with a description of the online model.

Online 2-Sided Myopic Template

while not empty( $\mathcal{N}$ )

$next :=$  lowest index (determined by adversary) of items in  $\mathcal{N}$

**Relevant Query:** Perform any set of relevant queries and update the *internal memory*

**Decision:** As a function of the *internal memory*, irrevocably accept or reject  $u_{next}$ , and remove  $u_{next}$  from  $\mathcal{N}$

A *priority function* is any injection  $\pi : Q(u) \rightarrow \mathbb{R}$ , where  $Q(u)$  is the vector of item  $u$ ’s marginals accessible under the appropriate relevant query model. The priority function determines the ordering of the input items. Specifically, at each iteration the item  $u_{next} \in \mathcal{N}$  with the highest priority (*i.e.*  $u_{next} =$

$\text{argmin}_{v \in \mathcal{N}}[\pi(Q(v))]$  is given to the algorithm. This leads to the more general models of fixed and adaptive priority algorithms.

#### Fixed Priority 2-Sided Myopic Template

**Ordering:** Specify a priority function  $\pi : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$   
while not empty( $\mathcal{N}$ )

*next* := index  $i$  of the item in  $\mathcal{N}$  that minimizes  $\pi(\rho(u_i|\emptyset), \bar{\rho}(u_i|\emptyset))$

**Relevant Query:** Using  $u_{next}$  as the next item, perform any set of relevant queries and update the *internal memory*

**Decision:** As a function of the *internal memory*, irrevocably accept or reject  $u_{next}$ , remove  $u_{next}$  from  $\mathcal{N}$  and update internal memory.

#### Adaptive Priority 2-Sided Myopic Template

while not empty( $\mathcal{N}$ )

**Ordering:** Given the *internal memory*, specify a priority function  $\pi$   
*next* := index  $i$  of the item in  $\mathcal{N}$  that minimizes  $\pi(Q(u_i))$

**Relevant Query:** Using  $u_{next}$  as the next item, perform any set of relevant queries and update the *internal memory*

**Decision:** As a function of the *internal memory*, irrevocably accept or reject  $u_{next}$ , remove  $u_{next}$  from  $\mathcal{N}$  and update internal memory.

A fixed priority algorithm determines  $\pi$  before it makes any oracle queries, and this ordering remains unchanged. For all relevant query models,  $Q(u)$  initially corresponds to the pair of  $u$ 's marginal gains in  $f$  and  $\bar{f}$  w.r.t. the empty set. Thus we write  $\pi : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ .

On the other hand, an adaptive priority algorithm may specify a new ordering function after each item is processed. Observe that in Q-Type 2 and 3, the length of  $Q$  increases with the iterations.

In both fixed and adaptive models, updating the internal memory also means deducing that certain marginal descriptions cannot exist and applying relevant queries (for the given  $Q$ -type) to obtain additional information. In particular, if  $u_{next} \in \mathcal{N}$  is the item with the minimum  $\pi$  value, then any item  $u_j$  with  $\pi(Q(u_j)) < \pi(Q(u_{next}))$  cannot appear later. We note that our inapproximability arguments result solely from information theoretic arguments. That is, the complexity or even computability of the ordering and decision steps is arbitrary, and there is no limitation on the size of the memory.

### 3 A 0.450 Inapproximation for Fixed Priority Algorithms

We design an LP whose solution gives the complete mapping of a submodular function  $f : 2^{\mathcal{N}} \rightarrow \mathbb{R}$ , over a small ground set  $\mathcal{N}$ . In the construction of  $f$ ,

we require certain items to be indistinguishable to the algorithm - allowing the adversary to control the input ordering. Denote by  $k$  the number of initial steps taken by the algorithm that the adversary will anticipate. That is, for  $2^k$  possible partial solutions, the adversary prepares an input ordering (consistent with the algorithm's queries in these  $k$  steps) such that any extendible solution has a bad approximation ratio. Since  $f$  may not have a succinct representation, we exemplify the adversarial strategy on a small MAX-DICUT instance, albeit with a worse lower bound.

**Theorem 4.** *For unweighted MAX-DICUT, no fixed order Q-Type 3 double-sided myopic algorithm can achieve an approximation ratio greater than  $\frac{2}{3}$ .*

*Proof.* Let  $v_1, \dots, v_6$  be the vertices along a directed 6-cycle  $G$  with unit edge weights. Clearly,  $OPT = 3$  is achieved by  $\{v_1, v_3, v_5\}$  or  $\{v_2, v_4, v_6\}$ . The regularity of  $G$  ensures that  $\pi(v_1) = \dots = \pi(v_6)$  for any  $\pi$ , allowing the adversary to specify any input ordering. Suppose the algorithm accepts (*resp.* rejects)  $v_1$  in the first step, the adversary fixes  $u_{i_2} = \{v_3, v_4\}$  as the next set of possible inputs in the sequence. At this point,  $v_3$  and  $v_4$  are indistinguishable to the algorithm, since  $\rho(v_3|S) = \rho(v_4|S) = 1$  and  $\bar{\rho}(v_3|S) = \bar{\rho}(v_4|S) = 1$  for any  $S \subseteq \mathcal{N}_1 = \{v_1\}$ . If the algorithm accepts  $u_{i_2}$ , then the adversary sets  $u_{i_2} = v_4$  (*resp.*  $u_{i_2} = v_3$ ); otherwise it sets  $u_{i_2} = v_3$  (*resp.*  $u_{i_2} = v_4$ ). Both cases contradict the optimal solutions, and the maximum cut value is now at most 2.  $\square$

### 3.1 Construction of the LP for Theorem 2

**Lemma 1.** *No fixed Q-Type 3 double-sided myopic algorithm can achieve an approximation ratio greater than  $\frac{1}{c}$  if there exists a (non-negative) submodular function  $f$  satisfying the following conditions:*

**Cond. 1**  $f(\{u\}) = f(\{v\}), \bar{f}(\{u\}) = \bar{f}(\{v\}), \forall u, v \in \mathcal{N}$ .

**Cond. 2** *There exist subsets  $A = \{a_1, \dots, a_k\} \subseteq \mathcal{N}, R = \{r_1, \dots, r_k\} \subseteq \mathcal{N}, A \cap R = \emptyset$ , such that for every  $0 < i < k$  and every  $\mathcal{C}_i \in \{a_1, r_1\} \times \{a_2, r_2\} \times \dots \times \{a_i, r_i\}$  and every subset  $S \subseteq \mathcal{C}_i$ ,*

$$\begin{aligned} f(S \cup \{a_{i+1}\}) &= f(S \cup \{r_{i+1}\}) \\ \bar{f}(S \cup \{a_{i+1}\}) &= \bar{f}(S \cup \{r_{i+1}\}) \end{aligned}$$

*Semantically,  $A$  (*resp.*  $R$ ) is the set of items that the algorithm is tricked into accepting (*resp.* rejecting). This is achievable if  $a_j, r_j$  are indistinguishable to the algorithm at round  $j$ , in the Q-Type 3 restricted oracle model.*

**Cond. 3** *For every  $\mathcal{C}_k \in \{a_1, r_1\} \times \{a_2, r_2\} \times \dots \times \{a_k, r_k\}$ , any solution  $S \subseteq \mathcal{N}$  such that  $\mathcal{C}_k \cap A \subseteq S$  and  $S \cap \mathcal{C}_k \cap R = \emptyset$  (i.e.  $S$  is an extension of  $\mathcal{C}_k$ ) must have  $f(S) \leq 1$ .*

**Cond. 4** *There exists a set  $S^* \in 2^{\mathcal{N}}$  such that  $f(S^*) \geq c$ .*

We formulate these conditions as an LP that maximizes  $c$ . For fixed  $n$  and  $k$ , define  $\mathcal{N} = \{s_1, \dots, s_{\lfloor \frac{n}{2} \rfloor}, o_1, \dots, o_{\lceil \frac{n}{2} \rceil}\}$  as the ground set, and designate

$O = \{o_1, \dots, o_{\lceil \frac{n}{2} \rceil}\}$  as the optimal solution. Set  $A = \{s_1, \dots, s_k\}$  and  $R = \{o_1, \dots, o_k\}$  to deter the algorithm from  $O$  as much as possible. For every  $S \subseteq \mathcal{N}$ , we abuse notation and refer to  $f(S)$  directly as the LP variable corresponding to the value of  $f$  on  $S$ . As this construction entails exponentially many variables, this is indeed only feasible in practice when restricted to a small  $\mathcal{N}$ . For interpretability, we may refer to the variable  $\bar{f}(S_i)$  as alias for the variable  $f(\mathcal{N} \setminus S_i)$  (following the definition of  $\bar{f}$ ). Define the linear program as follows:

**Objective**

$$\max f(\{o_1, \dots, o_{\lceil \frac{n}{2} \rceil}\}) \tag{Obj}$$

**Constraints**

$$f(S \cup \{v\}) + f(S \cup \{u\}) - f(S \cup \{v, u\}) \geq f(S) \quad \forall S \subseteq \mathcal{N}, (v, u) \in \mathcal{N} \setminus S \tag{Sub}$$

$$\begin{aligned} f(\{v\}) - f(\{u\}) &= 0, \\ \bar{f}(\{v\}) - \bar{f}(\{u\}) &= 0 \quad \forall v, u \in \mathcal{N} \tag{C1} \\ f(S \cup \{a_{i+1}\}) - f(S \cup \{r_{i+1}\}) &= 0, \\ \bar{f}(S \cup \{a_{i+1}\}) - \bar{f}(S \cup \{r_{i+1}\}) &= 0 \quad \forall S \subseteq \mathcal{C}_i, \end{aligned}$$

$$\begin{aligned} \forall \mathcal{C}_i \in \{a_1, r_1\} \times \dots \times \{a_i, r_i\}, \\ 0 < i < k \tag{C2} \end{aligned}$$

$$\begin{aligned} f(S) \leq 1 \quad \forall S \text{ s.t. } \exists \mathcal{C}_k, \mathcal{C}_k \cap A \subseteq S \\ \wedge S \cap \mathcal{C}_k \cap R = \emptyset \tag{C3} \end{aligned}$$

$$f(S) \geq 0 \quad \forall S \tag{C4}$$

$$f(\emptyset) = 0 \tag{C5}$$

Inequality (Sub) is a necessary and sufficient condition for submodularity [20], and (C4) and (C5) constrain  $f$  to be non-negative and normalized. The remaining constraints correspond to conditions 1-3 of Lemma 1. If the LP is feasible, then the objective value in (Obj) is a lower bound for  $c$  in condition 4.

*Proof of Theorem 2.* Using  $n = 8$  and  $k = 4$ , the LP described above produces a solution of  $c = 2.2222$ . By Lemma 1, this demonstrates a  $\frac{1}{2.2222} \approx 0.450$  inapproximation.  $\square$

## 4 A 0.432 Inapproximation for Adaptive Priority Algorithms

**Lemma 2.** *No Q-Type 2 adaptive double-sided myopic algorithm can achieve an approximation ratio greater than  $\frac{1}{c}$  if there exists a (non-negative) submodular function  $f$  that satisfies **Cond. 3** and **4** of Lemma 1, as well as the following:*



**Cond. 2\*** *There exist subsets  $A = \{a_1, \dots, a_k\} \subseteq \mathcal{N}$ ,  $R = \{r_1, \dots, r_k\} \subseteq \mathcal{N}$ ,  $A \cap R = \emptyset$ , such that for every  $i < k$  and every  $\mathcal{C}_i \in \{a_1, r_1\} \times \{a_2, r_2\} \times \dots \times \{a_i, r_i\}$  and all pairs  $v, u \in \mathcal{N} \setminus \mathcal{C}_i$ ,*

$$f((\mathcal{C}_i \cap A) \cup \{u\}) = f((\mathcal{C}_i \cap A) \cup \{v\})$$

$$\bar{f}((\mathcal{C}_i \cap R) \cup \{u\}) = \bar{f}((\mathcal{C}_i \cap R) \cup \{v\})$$

**Cond. 2\*** is the analog of **Cond. 2** of Lemma 1. Indistinguishability w.r.t. the weaker Q-Type 2 queries is now imposed on *all* remaining items in the first  $k$  steps (subsuming **Cond. 1**). As in Section 3.1, this leads to an LP representation.

*Proof of Theorem 3.* Using  $n = 8$  and  $k = 4$ , the LP described above produces a solution of  $c = 2.3158$ , giving us an inapproximability of  $\frac{1}{c} \approx 0.432$ .  $\square$

## 5 Further Discussion of the Double-Sided Myopic Model

Adapting the priority framework [3], we define the class of double-sided myopic algorithms and show the Buchbinder *et al.* algorithm can be realized as an online double-sided myopic algorithm. Similar to Poloczek’s [23] MAX-SAT inapproximation, our inapproximation results for deterministic double-sided myopic algorithms provide evidence that the randomized  $\frac{1}{2}$ -approximation double greedy for USM cannot be de-randomized even if the algorithms allow reasonable input orderings beyond the online constraint. Our double-sided interpretation of the greedy algorithm of [4] satisfies the deterministic model of Bar-Noy and Lampis [2], for which they give a  $\frac{2}{3\sqrt{3}}$  online inapproximability for MAX-DICUT.

Allowing randomized decisions, our myopic model also captures the oblivious algorithms of Feige and Jozeph [9] for the MAX-DICUT problem. Specifically, they define an oblivious algorithm as an online randomized algorithm that independently accepts each vertex  $v$  as a randomized function of the bias of the vertex, where  $bias(v) = \frac{w_{in}(v)}{w_{in}(v)+w_{out}(v)}$ . We note that the  $w_{in}$  (respectively,  $w_{out}$ ) values, as defined by [9], are precisely the values of  $\bar{\rho}(v|\emptyset)$  (resp.  $\rho(v|\emptyset)$ ). Hence, the bias of every node can be determined within our Q-Type 2 model. In fact, we could have included the initial marginals  $\bar{\rho}(v|\emptyset)$  and  $\rho(v|\emptyset)$  within the Q-Type 1 model as all our inapproximations ensure that all initial marginals are identical. We could have also introduced a Q-Type 0 model allowing only access to the initial marginals which suffices to model oblivious algorithms as randomized online myopic algorithms. However, to capture the Feige and Jozeph inapproximation bound, we would have to further restrict our algorithms so that decisions are not memory dependent. We note that the Paul *et al.* [21] derandomization of the Feige and Jozeph oblivious algorithm results in an online non-oblivious algorithm that goes beyond our myopic framework as it is specific to a graph input model where each vertex is represented by its bias as well as the bias of its neighbors. The myopic framework does not have access to individual edges and hence cannot determine the adjacent nodes.

We introduced a hierarchy of query types (respectively priority orderings) to illustrate the restricted nature of the Buchbinder *et al.* algorithm (Q-Type

1 and online) in contrast to more general input and priority models which can reasonably be said to follow the same double sided greedy approach introduced by Buchbinder *et al.* We believe that Q-Type 3 is perhaps the most general value oracle model that one can allow in a myopic algorithm. The fact that Q-Type 3 allows exponential time and memory only strengthens the inapproximation results. From a positive point of view, an efficient myopic algorithm would only make a small (e.g. linear or polynomial) number of oracle calls but would have exponentially many from which to choose. Q-Type 2 provides a natural way to restrict algorithms to polynomially many query calls.

Our inapproximations follow from an LP formulation of possible algorithmic decisions, and at present does not yield a succinctly defined problem. However, we provide a  $\frac{2}{3}$ -inapproximation for MAX-DICUT for fixed priority double-sided myopic algorithms. We again emphasize the generality of the myopic framework as it allows very general input orderings without imposing greediness in the decisions (as to rejecting or accepting an input). We also observe that *non-greediness* appears to be essential in both the randomized double-greedy algorithm of Buchbinder *et al.* as well as the deterministic approximation of Bar-Noy and Lampis for MAX-DICUT on DAGs; in contrast, the Buchbinder *et al.* deterministic algorithm does make greedy decisions.

**Acknowledgments.** The authors thank Yuval Filmus for the idea of employing LP, and Matthias Poloczek and Charles Rackoff for their helpful suggestions.

## References

1. Alon, N., Bollobás, B., Gyárfás, A., Lehel, J., Scott, A.: Maximum directed cuts in acyclic digraphs. *Journal of Graph Theory* **55**, 1–13 (2007)
2. Bar-Noy, A., Lampis, M.: Online maximum directed cut. *Journal of Combinatorial Optimization* **24**, 52–64 (2012)
3. Borodin, A., Nielsen, M.N., Rackoff, C.: (Incremental) priority algorithms. In: *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002*, pp. 752–761. Society for Industrial and Applied Mathematics, Philadelphia (2002)
4. Buchbinder, N., Feldman, M. Naor, J.S., Schwartz, R.: A tight linear time (1/2)-approximation for unconstrained submodular maximization. In: *Proceedings of the 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012*, pp. 649–658. IEEE Computer Society, Washington, DC (2012)
5. Calinescu, G., Chekuri, C., Pál, M., Vondrák, J.: Maximizing a submodular set function subject to a matroid constraint. In: *Integer Programming and Combinatorial Optimization*, pp. 182–196. Springer (2007)
6. Cornuejols, G., Fisher, M., Nemhauser, G.L.: On the uncapacitated location problem. In: Hammer, B.K.P.L., Johnson, E.L., Nemhauser, G. (eds.) *Studies in Integer Programming. Annals of Discrete Mathematics*, vol. 1, pp. 163–177. Elsevier (1977)
7. Cornuejols, G., Fisher, M.L., Nemhauser, G.L.: Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms. *Management Science* **23**, 789–810 (1977)
8. Feige, U.: A threshold of  $\ln n$  for approximating set cover. *J. ACM* **45**, 634–652 (1998)

9. Feige, U., Jozeph, S.: Oblivious algorithms for the maximum directed cut problem, arXiv preprint [arXiv:1010.0406](https://arxiv.org/abs/1010.0406) (2010)
10. Feige, U., Mirrokni, V.S., Vondrak, J.: Maximizing non-monotone submodular functions. In: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007, pp. 461–471. IEEE Computer Society, Washington, DC (2007)
11. Feldman, M., Naor, J.S., Schwartz, R.: Nonmonotone submodular maximization via a structural continuous greedy algorithm. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 342–353. Springer, Heidelberg (2011)
12. Filmus, Y., Ward, J.: A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In: 2012 IEEE 53rd Annual Symposium on Foundations of Computer Science (FOCS), pp. 659–668. IEEE (2012)
13. Gharan, S.O., Vondrák, J.: Submodular maximization by simulated annealing. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1098–1116. SIAM (2011)
14. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* **42**, 1115–1145 (1995)
15. Halperin, E., Zwick, U.: Combinatorial approximation algorithms for the maximum directed cut problem. In: Proc. of 12th SODA, pp. 1–7 (2001)
16. Huang, N., Borodin, A.: Bounds on double-sided myopic algorithms for unconstrained non-monotone submodular maximization, arXiv preprint [arXiv:1312.2173](https://arxiv.org/abs/1312.2173) (2013)
17. Kempe, D., Kleinberg, J.M., Tardos, É: Maximizing the spread of influence through a social network. In: KDD, pp. 137–146 (2003)
18. Lee, J., Mirrokni, V.S., Nagarajan, V., Sviridenko, M.: Non-monotone submodular maximization under matroid and knapsack constraints In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 323–332. ACM (2009)
19. Nemhauser, G.L., Wolsey, L.A.: Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research* **3**, 177–188 (1978)
20. Nemhauser, G.L., Wolsey, L.A., Fisher, M.L.: An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming* **14**, 265–294 (1978)
21. Paul, A.: Unconstrained submodular maximization priority algorithms. Cornell University (unpublished manuscript)
22. Paul, A., Poloczek, M., Williamson, D.: Priority algorithms for MAX DICUT. Cornell University, Unpublished manuscript (January 1, 2014)
23. Poloczek, M.: Bounds on greedy algorithms for MAX SAT. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 37–48. Springer, Heidelberg (2011)
24. Sviridenko, M.: A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters* **32**, 41–43 (2004)

# **Data Structures and Algorithms II**

# Tradeoff Between Label Space and Auxiliary Space for Representation of Equivalence Classes

Hicham El-Zein<sup>1</sup>(✉), J. Ian Munro<sup>1</sup>, and Venkatesh Raman<sup>2</sup>

<sup>1</sup> Cheriton School of Computer Science, University of Waterloo,  
Waterloo, ON N2L 3G1, Canada  
{helzein, imunro}@uwaterloo.ca

<sup>2</sup> The Institute of Mathematical Sciences, Chennai 600 113, India  
vraman@imsc.res.in

**Abstract.** Given a set of  $n$  elements that are partitioned into equivalence classes, we study the problem of assigning unique labels to these elements in order to support the query that asks whether the elements corresponding to two given labels belong to the same equivalence class. This problem has been studied by Katz et al [11], Alstrup et al [1], and Lewenstein et al [12]. Lewenstein et al [12] showed that if the labels were to be assigned from the set  $\{1, \dots, n\}$ , a data structure of size  $\Theta(\sqrt{n})$  bits is necessary and sufficient to represent the equivalence classes. They also showed that with no auxiliary data structure, a label space of size  $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$  is necessary and sufficient. Our main result is that if we allow a label space of size  $cn$  for any constant  $c > 1$ , a data structure of size  $\Theta(\log n)$  bits is necessary and sufficient. We also show that the equivalence query in such a data structure can be answered in  $\Theta(1)$  time. We believe that our work can trigger further work on tradeoffs between label space and auxiliary data structure space for other labeling problems.

## 1 Introduction and Motivation

Given a partition of an  $n$  element set into equivalence classes, our problem is to preprocess the set, assigning a unique label to each element, to obtain a data structure with minimum space to support the following query: given two labels, determine whether their corresponding elements are in the same equivalence class. We call such queries ‘equivalence queries’. This is a fundamental data structures problem and it has various applications such as testing whether two vertices are in the same connected component in an undirected graph or in the same strongly connected component in a directed graph. We study the problem from the perspective of succinct data structures. Our aim is to develop data structures whose size is within a constant factor of the information theoretic lower bound. Designing succinct data structures is an area of interest in theory and practice motivated by the need of storing large amount of data using

---

This work was sponsored by the NSERC of Canada and the Canada Research Chairs Program.

the smallest space possible. For succinct representations of dictionaries, trees, arbitrary graphs and partially ordered sets see [2–8, 14–16]

Kannan, Naor and Rudich [10] were the first to introduce the concept of labeling schemes to answer graph adjacency queries. Katz, Katz, Korman and Peleg [11] extended this notion to support graph flow and connectivity queries. They studied the problem of assigning (not necessarily distinct) labels to graph nodes so that queries can be answered by just looking at the labels of the queried nodes. They showed that to answer  $k$ -connectivity queries the length of each label has a lower bound of  $\Omega(k \log n)$  bits. For the problem that is considered in this paper  $k = 1$ , their lower bound implies a  $\lceil \log n \rceil$  lower bound for the length of each label. However, in some situations we may want to distinguish between individual nodes within the same component, so we may want to give unique labels to each node. Lewenstein et al [12] studied this problem in two models:

- In the first model, the query is to be answered by just examining the labels of the queried elements. They called this problem the *direct equivalence queries* problem. They tightened the lower bound in [1] to a label space of  $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$ , which can be represented in  $\log n + \log \log n - \Omega(1)$  bits, and proved that it is sufficient. Moreover, they showed that one can solve the problem using  $\log n + \log \log n + 2$  bits such that equivalence queries can be answered in  $\Theta(1)$  time.
- In the second model, the  $n$  elements are to be assigned unique labels from the set  $\{1, \dots, n\}$ . They showed that an auxiliary data structure of size  $\Theta(\sqrt{n})$  bits is necessary and sufficient to represent the equivalence class information. They supported the query in such a structure in  $\Theta(\log n)$  time. Moreover, they developed structures where queries can be answered:
  - in  $O(\log \log n)$  time using  $O(\sqrt{n} \log n / \log \log n)$  bits, and
  - in  $O(1)$  time using  $O(\sqrt{n} \log n)$  bits of space.

Our first observation in this paper is that one can combine their data structures to obtain a structure that uses  $O(\sqrt{n} \log n / f(n))$  bits of space, and answer queries in  $O(f(n))$  time for any  $f(n)$  in  $O(\log n)$ . We also notice an inversely proportional relation between label space and auxiliary data structure size. If the label space is in the range of  $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$ , no data structure is required. Once the label space range is decreased to  $n$ , a data structure of size  $O(\sqrt{n})$  bits is necessary. In this paper, we further investigate this relation. Our work is motivated by the fact that unless  $n$  is a power of two or a little less than a power of two, we can increase the label space by a constant factor without increasing the number of bits required to store each address. To be more specific, we investigate the case where the labels are to be assigned from the set  $\{1, \dots, cn\}$  where  $c > 1$  ( $c$  can be a decimal less than 2). We show that an auxiliary data structure whose size is  $\Theta(\log n)$  bits is necessary and sufficient to represent the equivalence class information in this case. Such a structure can be completely stored in cache memory. Moreover, we can support the query in such a structure in  $\Theta(1)$  time.

The rest of this paper is divided as follows. In Section 2, we present some basic definitions. In Section 3, we provide the scheme that unites the three data structures presented in [12]. In Section 4 we present our data structure to

represent an equivalence class using  $O(\log n)$  bits once we allow the label space to be  $cn$  for some  $c > 1$ . In Section 5, we show that  $\Omega(\log n)$  bits are necessary with such a label space. In section 6, we give a data structure that represent an equivalence class using  $O(\log n / \log f(n))$  bits once we allow the label space to be  $f(n) \cdot n$  where  $f(n) \in \omega(1)$  and  $f(n) \in O(\log(n))$ . Finally, we conclude our work in Section 6.

## 2 Definitions

In this section, we briefly describe the required definitions and backgrounds. An *integer partition*  $p$  of  $n$  is a multiset of positive integers that sum to  $n$ . We call these positive integers the *classes* of  $p$ , and we denote by  $|p|$  this number of classes. We say that a partition  $p$  of  $n$  *dominates* a partition  $q$  of  $m$  where  $n > m$  if  $q$  *fits* in  $p$ . To be more precise  $p$  dominates  $q$  if:

- $|p| \geq |q|$  and
- for  $1 \leq i \leq |q|$ , the  $i^{\text{th}}$  largest class (breaking ties arbitrarily) in  $p$  is at least as big as the  $i^{\text{th}}$  largest class in  $q$ .

For example, the partition  $\{7, 7, 6\}$  of 20 dominates the partition  $\{5, 5\}$  of 10, but not the partition  $\{8, 2\}$  of 10. Given a partition  $p$  of  $n$ , we define a *part*  $q$  of length  $k$  to be a collection of classes in  $p$  that sum to  $k$ . We say that a size  $s$  *fills*  $q$  if  $q$  contains  $\lfloor k/s \rfloor$  classes of size  $s$  and a class of size  $k \bmod s$ . Finally we say that two parts *intersect* if they share a common class; otherwise, they are *non-intersecting*.

## 3 Time-Space Tradeoffs with Label Space $[1, 2, \dots, n]$

We assume that the equivalence class is given by a tuple containing the sizes of the classes, and our task is to give each element a unique label from the range 1 to  $n$ .

Towards the end of this section, based on the work in [12], we show that for any function  $f(n) = O(\log n)$  there exists a data structure using  $O(\sqrt{n} \log n / f(n))$  bits that has  $O(f(n))$  query time. First, we revise the main data structure of [12].

The number of partitions of an  $n$  element set into equivalence classes is the same as the number of integer partitions of  $n$ , which by the Hardy-Ramanujan formula [9] is asymptotically equivalent to  $\frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$ . Thus the information theoretic lower bound for representing the equivalence class relation is  $\Theta(\sqrt{n})$ .

Let  $k$  be the number of distinct class sizes. For  $i = 1$  to  $k$ , let  $s_i$  be the distinct sizes of the classes, and let  $n_i$  be the number of classes of size  $s_i$ . Order the classes in non-decreasing order of  $\gamma_i = s_i n_i$  so that for  $i = 1$  to  $k - 1$ ,  $s_i n_i \leq s_{i+1} n_{i+1}$ . Notice that since  $\sum_{i=1}^k s_i n_i = n$ ,  $k$  is bounded by  $\sqrt{2n}$ . The primary data structure is made up of two sequences:

- the sequence  $s$  that consists of  $\delta_1 = s_1 n_1$  and  $\delta_i = s_i n_i - s_{i-1} n_{i-1}$ , for  $i = 2$  to  $k$  and
- the sequence  $m$  that consists of  $n_i$ , for  $i = 1$  to  $k$ .

Elements of the two sequences are represented in binary. Since the length of each element may vary, we store two other sequences that shadow the primary sequences. The shadow sequences have a 1 at the starting point of each element in the shadowed sequence and a 0 elsewhere. Also store a select structure (see [13] or [8, 16]) on the two shadow sequences in order to identify the 1s quickly. It is shown in [12] that the space occupied by these sequences is  $O(\sqrt{n})$ .

Assign labels to elements based on the first sequence. To be more specific, for the classes of size  $s_i$ , assign label values from  $\sum_{j=1}^{i-1} s_j n_j + 1$  to  $\sum_{j=1}^i s_j n_j$ . Next, we show how to implement the equivalence query. Given an element labeled  $x$ , first find the predecessor  $p(x)$  of  $x$ , which is  $\max\{j \mid \sum_{i=1}^j s_i n_i < x\}$ . Given  $x$  and  $y$ , if  $p(x) \neq p(y)$  then  $x$  and  $y$  belong to different equivalence classes. If  $p(x) = p(y)$ , then we know that  $x$  and  $y$  belong to equivalence classes of the same size. They are in the same equivalence class if and only if  $\lceil (x - \sum_{i=1}^{p(x)} s_i n_i) / n_{p(x)+1} \rceil$  is equal to  $\lceil (y - \sum_{i=1}^{p(y)} s_i n_i) / n_{p(y)+1} \rceil$ . Using the select data structures saved we can find the value of  $n_i$  in constant time, so what is left is answering the predecessor query.

Given any function  $f(n) = O(\log n)$  we can store the values of  $\sum_{j=1}^i s_j n_j$  for each  $i$  that is a multiple of  $f(n)$  in a fully indexable dictionary [16] with the improved redundancy of [8] (which supports the predecessor query in constant time) using  $O((\sqrt{n}/f(n))^{1+\epsilon})$  bits. However due to Lemma 2 in [12] this can be done in  $O(\sqrt{n} \log n / f(n))$  bits. Now  $p(x)$  can be obtained by doing a constant time look up on the partial sum values, then the actual predecessor can be found by doing a linear search on the delta values in this range.

The following theorem gives a uniform treatment for the three techniques of [12], and provides a generic trade-off between the auxiliary data structure size and query time.

**Theorem 1.** *Given a partition of an  $n$  element set into equivalence classes and a function  $f(n) = O(\log n)$ ,  $O(\sqrt{n} \log n / f(n))$  bits are sufficient for storing the partition and to answer the equivalence query in  $O(f(n))$  time if each element is to be given a unique label in the range  $\{1, 2, \dots, n\}$ .*

- By setting  $f(n) = \log n$ , we get a structure using  $O(\sqrt{n})$  bits that can answer the equivalence query in  $O(\log n)$  time.
- By setting  $f(n) = \log \log n$ , we get a structure using  $O(\sqrt{n} \log n / \log \log n)$  bits that can answer the equivalence query in  $O(\log \log n)$  time.
- By setting  $f(n) = 1$ , we get a structure using  $O(\sqrt{n} \log n)$  bits that can answer the equivalence query in  $O(1)$  time.

## 4 Succinct Data Structures with Label Space $[1, 2, \dots, cn]$

In this section, we move on to designing data structures where the  $n$  elements can be freely labelled with unique labels in the range of 1 to  $cn$ . Our work is



motivated by the fact that unless  $n$  is a power of two or a little less than a power of two, we can increase the address space by a constant factor without increasing the number of bits required to store each address. The queries can be answered by looking at an auxiliary data structure. We are interested in time and space efficient data structures that are within a constant factor from the information theoretic lower bound. We first assign an implicit ordering of the elements. Each element gets a label according to this ordering, and the queries are answered by looking at these labels and the auxiliary data structure. Theorem 2 and 3 are to lead the reader to our main result in this section which is Theorem 4.

**Theorem 2.** *Given a partition of an  $n$  element set into equivalence classes,  $O(\log^2 n)$  bits are sufficient for storing the partition and to answer the equivalence query in  $O(1)$  time if the elements are to be given unique labels in the range  $\{1, 2, \dots, cn\}$  for any constant  $c > 1$ .*

*Proof.* Given a partition, first round the size of each class to the nearest power of  $c$ . The required address space will increase from  $n$  to at most  $cn$  because the size of each class increased by at most a factor of  $c$ . As in Section 2, let  $k$  be the number of distinct class sizes (note that  $k \leq \log_c(n)$ ). For  $i = 1$  to  $k$ , let  $s_i$  be the distinct sizes of the classes, and let  $n_i$  be the number of classes of size  $s_i$ . Order the classes in non-decreasing order of  $\gamma_i = s_i n_i$  so that for  $i = 1$  to  $k - 1$ ,  $s_i n_i \leq s_{i+1} n_{i+1}$ . Assign labels to elements such that classes of size  $s_i$  are assigned values from  $\sum_{j=1}^{i-1} s_j n_j + 1$  to  $\sum_{j=1}^i s_j n_j$ . Our data structure consists of:

- a fully indexable dictionary [16] with the improved redundancy of [8] that supports the predecessor query in constant time, storing the values  $\sum_{j=1}^i s_j n_j$  for  $i = 1$  to  $k$ . This structure uses  $\Theta(\log^2 n)$  bits or  $\Theta(\log n)$  words (see Lemma 2 in [12]), and
- a sequence that consists of  $n_i$ , for  $i = 1$  to  $k$ .

The equivalence query can be answered in  $O(1)$  time using the method described in Section 3. □

To further reduce the size of the data structure, round the size of each class to the nearest power of  $\sqrt{c}$ , then round each  $n_i$  value to the nearest power of  $\sqrt{c}$ . These operations will increase the address space by at most a factor of  $(\sqrt{c})^2 = c$ . To store the equivalence relation, it is sufficient to store the logarithm of the  $n_i$  values, which can be done in  $O(\log n \log \log n)$  bits.

**Theorem 3.** *Given a partition of an  $n$  element set into equivalence classes,  $O(\log n \log \log n)$  bits are sufficient for storing the partition and to answer the equivalence queries in constant time if each element has to be given a unique label in the range  $\{1, 2, \dots, cn\}$  for any constant  $c > 1$ .*

Finally we describe how to obtain a data structure whose size is  $O(\log n)$  bits. First we round the size of each class to the nearest power of  $\sqrt{c}$ , increasing the label space by at most a factor of  $\sqrt{c}$ . Next, we make sure that the distinct class sizes fill non-intersecting parts whose length is a multiple of  $s = \lfloor kn/(\lceil \log_{\sqrt{c}}(n) \rceil) \rfloor$  where  $k = c - \sqrt{c}$ . Since we have at most  $\lceil \log_{\sqrt{c}}(n) \rceil$  distinct class sizes, this operation will increase our address space by at most  $kn$ , so the new address space will have an upper bound of:

$$\begin{aligned} n(\sqrt{c} + k) &= n(\sqrt{c} + (c - \sqrt{c})) \\ &= cn \end{aligned}$$

as we desired. Let the length of the part filled by  $\lfloor (\sqrt{c})^i \rfloor$  be equal to  $c_i s$  (note that  $c_i$  can be equal to 0). Notice that:

$$\begin{aligned} \sum_{i=0}^{\lceil \log_{\sqrt{c}}(n) \rceil} c_i &\leq cn/s \\ &= c \lceil \log_{\sqrt{c}}(n) \rceil / k \\ &= O(\log(n)) \end{aligned}$$

To represent the equivalence class relation it is sufficient to store the  $c_i$  values. Our data structure consists of a single bit vector  $\psi$  that stores the  $c_i$  values in unary with a 0 separator between each two consecutive values. We also store a select structure on  $\psi$  to identify the 1s and 0s quickly, and we store a rank structure to count the 1s and 0s quickly. The space required is  $O((c/k + 1) \lceil \log_{\sqrt{c}}(n) \rceil) = O(\log n)$  bits.

Assign labels to elements such that classes of size  $\lfloor (\sqrt{c})^i \rfloor$  are assigned values from the range  $\sum_{j=0}^{i-1} c_j s$  to  $(\sum_{j=0}^i c_j s) - 1$ .

**Implementing the Equivalence Query.** Now given an element labeled  $x$ , we can determine the size of the equivalence class that  $x$  belongs to by getting the number of zeroes  $i$  before the  $\lfloor x/s \rfloor$ -st 1 in  $\psi$ . Once we find  $i$  we know that  $x$  belongs to an equivalence class of size  $\lfloor (\sqrt{c})^i \rfloor$ . Given two elements  $x$  and  $y$ , if they belong to classes with different sizes, then  $x$  and  $y$  are not in the same equivalence class.

If  $x$  and  $y$  both belong to a class of size  $\lfloor (\sqrt{c})^i \rfloor$ , then calculate  $j$  the number of 1s before the  $i$ -th 0 in  $\psi$ .  $x$  and  $y$  are in the same equivalence class if and only if  $\lfloor (x - js) / \lfloor (\sqrt{c})^i \rfloor \rfloor = \lfloor (y - js) / \lfloor (\sqrt{c})^i \rfloor \rfloor$ .<sup>1</sup>

**Theorem 4.** *Given a partition of an  $n$  element set into equivalence classes,  $O(\log n)$  bits are sufficient for storing the partition and to answer the equivalence query in  $O(1)$  time if each element is to be given a unique label in the range  $\{1, 2, \dots, cn\}$  for any constant  $c > 1$ .*

<sup>1</sup> We assume that we are working in a RAM Model where  $(\sqrt{c})^i$  can be computed in  $O(1)$  time. Relaxing this assumption would increase the query time to  $O(\log \log(n))$ .

### 5 Lower Bound

In this section, we show that the space bound of Theorem 4 is optimal for the range of label space used. Without loss of generality, to make our calculations easier, we assume that  $n$  is a power of 2.

Let  $S_{cn}$  be the set of all partitions of  $\lfloor cn \rfloor$  and  $S_n$  the set of all partitions of  $n$ . While one partition of  $\lfloor cn \rfloor$  can dominate many partitions of  $n$ , we argue first that at least  $(\log n)/2c$  partitions of  $\lfloor cn \rfloor$  are necessary to dominate all partitions of  $n$ . Let  $\mathcal{S}$  be the smallest set of partitions of  $\lfloor cn \rfloor$  that dominates all the partitions of  $n$ . Our first claim is that:

**Lemma 1.**  $|\mathcal{S}| \geq \log n/(2c)$

*Proof.* Consider the subset  $Q$  of  $S_n$  defined as follows:

- $Q$  contains  $\log n + 1$  partitions, and
- the  $i^{th}$  partition, for  $i = 0$  to  $\log n$ ,  $q_i$  of  $Q$  contains  $n/2^i$  classes of size  $2^i$ .

Let  $p$  be a partition of  $\lfloor cn \rfloor$  that dominates partitions  $q_{j_1}, q_{j_2}, \dots, q_{j_m}$  of  $Q$  where  $j_1 < j_2 < \dots < j_m$ .

To dominate  $q_{j_m}$ ,  $p$  must contain at least  $n/2^{j_m}$  classes of size  $2^{j_m}$ . Since  $p$  dominates  $q_{j_{i+1}}$ , for any  $1 \leq i < m$ , there must exist at least  $n/2^{j_{i+1}}$  classes of size greater than  $2^{j_i}$  in  $p$ . Therefore, for  $p$  to dominate  $q_{j_i}$ ,  $p$  must contain at least:

$$\begin{aligned} n/2^{j_i} - n/2^{j_{i+1}} &\geq n/2^{j_i} - n/2^{j_i+1} \\ &\geq n/2^{j_i+1} \end{aligned}$$

additional classes of size greater than or equal to  $2^{j_i}$ . Consequently:

$$\begin{aligned} cn &\geq n + \sum_{k=1}^{m-1} 2^{j_k} n/2^{j_k+1} \\ &\geq \sum_{k=1}^m n/2 \\ &\geq mn/2 \end{aligned}$$

and  $m \leq 2c$ . Thus any partition of  $\lfloor cn \rfloor$  can dominate at most  $2c$  partitions of  $Q$ , and to dominate  $Q$  we need a minimum of  $(\log n + 1)/(2c) = \Omega(\log n)$  partitions of  $\lfloor cn \rfloor$ . Since  $Q$  is a subset of  $S_n$ , our claim holds.  $\square$

Extending the above argument, we show

**Lemma 2.** Let  $k \geq 1$  be any integer such that  $\log(n/k) > 2ck$ , then  $|\mathcal{S}| \geq \binom{\log(n/k)}{k} / \binom{2ck}{k}$ .

*Proof.* Divide  $n$  into  $k$  parts each of size  $n/k$ . Let  $Q$  be the set formed by filling each part with a distinct power of 2, clearly  $|Q| = \binom{\log(n/k)}{k}$ .

Let  $p$  be a partition of  $\lfloor cn \rfloor$  such that  $p$  dominates  $m$  parts filled by the following distinct powers of 2:  $2^{j_1}, 2^{j_2}, \dots, 2^{j_m}$  where  $j_1 < j_2 < \dots < j_m$ .

To dominate the part filled by  $2^{j_m}$ ,  $p$  must contain at least  $n/(k2^{j_m})$  classes of size  $2^{j_m}$ . Since  $p$  dominates the part filled by  $2^{j_{i+1}}$ , for any  $1 \leq i < m$ , there must exist at least  $n/(k2^{j_{i+1}})$  classes of size greater than  $2^{j_i}$  in  $p$ . Therefore, for  $p$  to dominate the part filled by  $2^{j_i}$ ,  $p$  must contain at least:

$$\begin{aligned} n/(k2^{j_i}) - n/(k2^{j_{i+1}}) &\geq n/(k2^{j_i}) - n/(k2^{j_i+1}) \\ &\geq n/(k2^{j_i+1}) \end{aligned}$$

additional classes of size greater than or equal to  $2^{j_i}$ . Consequently:

$$\begin{aligned} cn &\geq n/k + \sum_{i=1}^{m-1} 2^{j_i} n/(k2^{j_i+1}) \\ &\geq \sum_{i=1}^m n/(2k) \\ &\geq mn/(2k) \end{aligned}$$

and  $m \leq 2ck$ . Thus any partition of  $\lfloor cn \rfloor$  can dominate at most  $2ck$  distinct parts, and any partition of  $\lfloor cn \rfloor$  can dominate at most  $\binom{2ck}{k}$  partitions of  $Q$ . Hence, to dominate  $Q$  we need a minimum of  $\binom{\log(n/k)}{k} / \binom{2ck}{k}$  partitions of  $\lfloor cn \rfloor$ . Since  $Q$  is a subset of  $S_n$  our claim holds.  $\square$

The information theoretic lower bound for space to represent the equivalence class information is given by  $\log(|S|) \geq \log(\binom{\log(n/k)}{k} / \binom{2ck}{k})$ , if we choose  $k = \log n/4c$  we get our desired bound  $\log(|S|) = \Omega(\log n)$ .

**Theorem 5.** *Given a partition of an  $n$  element set into equivalence classes,  $\Theta(\log n)$  bits are necessary and sufficient for storing the partition if each element is to be given a unique label in the range  $\{1, 2, \dots, cn\}$  for any constant  $c > 1$ . Moreover, the equivalence query in such a structure can be answered in  $O(1)$  time.*

## 6 Data Structure with Label Space $[1, 2, \dots, f(n) \cdot n]$

In this section, we generalize the techniques presented in section 4 to design a data structure where the  $n$  elements can be freely labelled with unique labels in the range of 1 to  $f(n)n$  where  $f(n) \in \omega(1)$  and  $f(n) \in O(\log(n))$ .

**Theorem 6.** *Given a partition of an  $n$  element set into equivalence classes and a function  $f(n)$  where  $f(n) \in \omega(1)$  and  $f(n) \in O(\log(n))$ ,  $O(\log(n)/\log(f(n)))$  bits are sufficient for storing the partition and to answer the equivalence query in  $O(1)$  time if the elements are to be given unique labels in the range  $\{1, 2, \dots, f(n)n\}$ .*

*Proof.* Let  $l = f(n)/2$ , we round the size of each class to the nearest power of  $l$ , increasing the label space by at most a factor of  $l$ . Then, we make sure that the distinct class sizes fill non-intersecting parts whose length is a multiple of  $s = \lfloor 2^{\lceil \log(n) \rceil} f(n) / (4 \lceil \log_l(n) \rceil) \rfloor$ . Since we have at most  $\lceil \log_l(n) \rceil$  distinct class sizes, this operation will increase our address space by at most  $2^{\lceil \log(n) \rceil} f(n)/4$ , so the new address space will have an upper bound of:

$$\begin{aligned} n(f(n)/2) + 2^{\lceil \log(n) \rceil} f(n)/4 &\leq n(f(n)/2) + 2nf(n)/4 \\ &= f(n)n \end{aligned}$$

as we desired. Let the length of the part filled by  $\lfloor (l)^i \rfloor$  be equal to  $c_i s$ . Notice that:

$$\begin{aligned} \sum_{i=0}^{\lceil \log_l(n) \rceil} c_i &\leq f(n)n/s \\ &\leq 4 \lceil \log_l(n) \rceil \\ &= O(\log(n)/\log(f(n))) \end{aligned}$$

To represent the equivalence class relation, we store the value of  $\lceil \log(n) \rceil$  and  $f(n)$  using  $O(\log \log(n))$  bits. Moreover, we store the  $c_i$  values in  $O(\log(n)/\log(f(n)))$  bits using the same method as in Theorem 4. To answer the equivalence query, we calculate the value of  $s$  and  $l$ , then we apply the same procedure as in Theorem 4.  $\square$

## 7 Conclusion

We have discussed the trade-off between label space and auxiliary space for the fundamental problem of supporting equivalence queries. Our main result is to show that once labels are assigned from the range  $cn$  a data structure whose size is  $\Theta(\log n)$  bits is necessary and sufficient to represent the equivalence classes. Our scheme allows an implicit labeling of elements and supports equivalence queries in  $O(1)$  time. The main motivation behind our work is that when  $n$  is not a power of 2 or slightly less than a power of 2, we can increase the address space without increasing the number of bits required to store each label. Thus, for most values of  $n$ , our result is achieved using an optimal number of bits, which is  $\lceil \log n \rceil$  bits.

Apart from providing, what we believe, a non-trivial data structure requiring only  $\Theta(\log n)$  bits, we have also touched upon the interesting tradeoff issue between auxiliary space and the label space. As there is a huge body of research in ‘labeling schemes’ (see [1]), investigation into such a tradeoff for other labeling schemes maybe interesting.

## References

1. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, Maryland, USA, January 12–14, pp. 689–698 (2003)

2. Barbay, J., Castelli Aleardi, L., He, M., Munro, J.I.: Succinct representation of labeled graphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 316–328. Springer, Heidelberg (2007)
3. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. *Algorithmica* **43**(4), 275–292 (2005)
4. Brodnik, A., Munro, J.I.: Membership in constant time and almost-minimum space. *SIAM Journal on Computing* **28**(5), 1627–1640 (1999)
5. Farzan, A., Munro, J.I.: Succinct representations of arbitrary graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 393–404. Springer, Heidelberg (2008)
6. Farzan, A., Munro, J.I.: A uniform paradigm to succinctly encode various families of trees. *Algorithmica* **68**(1), 16–40 (2014)
7. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11–14, pp. 1–10 (2004)
8. Grossi, R., Orlandi, A., Raman, R., Rao, S.S.: More haste, less waste: Lowering the redundancy in fully indexable dictionaries. In: Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, Freiburg, Germany, February 26–28, pp. 517–528 (2009)
9. Hardy, G.H., Ramanujan, S.: Asymptotic formulae in combinatory analysis. *Proceedings of the London Mathematical Society* **2**(1), 75–115 (1918)
10. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. *SIAM Journal on Discrete Mathematics* **5**(4), 596–603 (1992)
11. Katz, M., Katz, N.A., Korman, A., Peleg, D.: Labeling schemes for flow and connectivity. *SIAM Journal on Computing* **34**(1), 23–40 (2004)
12. Lewenstein, M., Munro, J.I., Raman, V.: Succinct data structures for representing equivalence classes. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) *Algorithms and Computation*. LNCS, vol. 8283, pp. 502–512. Springer, Heidelberg (2013)
13. Munro, J.I.: Tables. In: *Foundations of Software Technology and Theoretical Computer Science*, pp. 37–42. Springer (1996)
14. Munro, J.I., Nicholson, P.K.: Succinct posets. In: Epstein, L., Ferragina, P. (eds.) *ESA 2012*. LNCS, vol. 7501, pp. 743–754. Springer, Heidelberg (2012)
15. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses, static trees and planar graphs. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, October 19–22, pp. 118–126 (1997)
16. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)* **3**(4), 43 (2007)

# Depth-First Search Using $O(n)$ Bits

Tetsuo Asano<sup>1</sup>, Taisuke Izumi<sup>2</sup>, Masashi Kiyomi<sup>3</sup>(✉), Matsuo Konagaya<sup>1</sup>,  
Hiroataka Ono<sup>4</sup>, Yota Otachi<sup>1</sup>, Pascal Schweitzer<sup>5</sup>, Jun Tarui<sup>6</sup>,  
and Ryuhei Uehara<sup>1</sup>

<sup>1</sup> JAIST, Nomi, Japan

<sup>2</sup> Nagoya Institute of Technology, Nagoya, Japan

<sup>3</sup> Yokohama City University, Yokohama, Japan

masashi@yokohama-cu.ac.jp

<sup>4</sup> Kyushu University, Fukuoka, Japan

<sup>5</sup> RWTH Aachen University, Aachen, Germany

<sup>6</sup> University of Electro-Communications, Chofu, Japan

**Abstract.** We provide algorithms performing Depth-First Search (DFS) on a directed or undirected graph with  $n$  vertices and  $m$  edges using only  $O(n)$  bits. One algorithm uses  $O(n)$  bits and runs in  $O(m \log n)$  time. Another algorithm uses  $n + o(n)$  bits and runs in polynomial time. Furthermore, we show that DFS on a directed acyclic graph can be done in space  $n/2^{\Omega(\sqrt{\log n})}$  and in polynomial time, and we also give a simple linear-time  $O(\log n)$ -space algorithm for the depth-first traversal of an undirected tree. Finally, we also show that for a graph having an  $O(1)$ -size feedback set, DFS can be done in  $O(\log n)$  space. Our algorithms are based on the analysis of properties of DFS and applications of the  $s$ - $t$  connectivity algorithms due to Reingold and Barnes et al., both of which run in sublinear space.

## 1 Introduction

Computation with *limited memory* has been a quite active topic of research recently. Partly motivated by the massive data phenomenon in practice, classical computational problems have been reconsidered in terms of various streaming models, for example. This paper is concerned with the more classical Random Access Machine (RAM) model, where input data is in read-only random access memory, and computation proceeds using additional working space, which, for example, consists of  $O(\log n)$  or  $o(n)$  or  $O(n)$  bits. The output will be stored in a write-only output tape. For this model, recent works have given some new interesting memory-limited algorithms: Elberfeld et al. [7] and Elberfeld and Kawarabayashi [8] have given  $O(\log n)$ -space algorithms for solving a family of fundamental graph problems (more precisely those problems expressibly in monadic second-order language on graphs of bounded tree-width) and for the canonization of graphs of bounded genus. Very recently, Asano et al. [5] and Imai et al. [9] have shown that the reachability problem on directed graphs can be solved using only  $\tilde{O}(\sqrt{n})$  space for planar graphs.

In this paper, we investigate how one can perform Depth-First Search (DFS) of a given graph using limited amount of memory. DFS, being one of the most fundamental and important ways to search or explore a graph, is used as a subroutine in many prominent graph algorithms [15]. A better understanding of DFS in terms of space complexity and memory-efficient algorithms is desirable, but it appears that such aspects of DFS have not been considered much. This is perhaps because the problem is trivial if one uses  $O(n \log n)$  bits of memory on one hand, where  $n$  is the number of vertices of a graph, and, on the other hand, the problem is P-complete [13] and thus polylogarithmic-space algorithms are unlikely to exist.

## 1.1 The DFS Problem

Before we outline previous work on DFS, we explain some technical details about the DFS problem. We can cast a DFS problem in various ways. The output can be (1) the DFS tree, or the output can be (2) the DFS numbering of each vertex, that is, the ordering of vertices with respect to the time of the first visit, or (3) the input can be a graph together with two vertices  $u$  and  $v$ , and the output can be the yes/no answer as to whether vertex  $u$  is visited before vertex  $v$  in DFS. For our purposes, which of the three variants above we consider does not matter since they can all be reduced to each other using  $O(\log n)$  space. Furthermore, all the algorithms we present can directly handle any of the three variants in a straightforward way. For definiteness, we think of DFS problem as the DFS tree construction problem.

We assume that an input graph is given by an adjacency list. Suppose that DFS is visiting a vertex  $v$  for the first time, reaching  $v$  from vertex  $u$ . DFS will now visit the first unvisited neighbor of  $v$ , where the “first” is usually with respect to either one of the following two orders: (1) the appearance order in  $v$ ’s adjacency list; or, (2) in the case of undirected graphs: under the assumption that  $n$  vertices are numbered  $1, \dots, n$  and with respect to the cyclic ordering of  $1, \dots, n$ , the unvisited vertex  $x$  among  $v$ ’s neighbors that appears first after  $u$  in the cyclic ordering.

DFS with respect to either one of the two scenarios above is sometimes called *lexicographically smallest DFS* or *lexicographic DFS* or *lex-DFS* [16], [17] (and sometimes simply called DFS). Usually, a lex-DFS algorithm can handle both scenarios (1) and (2) in the same manner, and one does not need to distinguish the two scenarios. All algorithms in this paper perform lex-DFS.

In contrast to lex-DFS, we can also consider an algorithm that outputs *some* DFS tree of a given graph. Such an algorithm treats an adjacency list as a *set*, ignoring the order of appearance of vertices in it, and outputs a spanning tree  $T$  such that there exists *some* adjacency ordering  $R$  such that  $T$  is the DFS tree with respect to  $R$ . We say that such a DFS algorithm performs *general-DFS*.

## 1.2 Related Work

Reif [13] has shown that lex-DFS is P-complete. Anderson and Mayr [2] have shown that computing the lexicographically first maximal path, that is, computing the leftmost root-to-leaf path of the lex-DFS tree, is already P-complete.



Aggarwal and Anderson [1] have shown that general-DFS is computable in RNC, that is, computable by a randomized parallel algorithm with polynomially many processors and in polylogarithmic parallel time in the PRAM model, or, equivalently, by randomized polynomial-size poly-logarithmic depth circuits. There is no known deterministic NC algorithm for general-DFS.

In a seminal work, Reingold [14] has given a deterministic  $O(\log n)$ -space algorithm for the Undirected  $s$ - $t$  Connectivity Problem:

**Theorem 1 (Reingold [14]).** *Given an undirected graph and two vertices  $s$  and  $t$ , determining whether  $s$  and  $t$  are connected can be done in deterministic  $O(\log n)$  space.*

Using Reingold's algorithm, one can compute a minimum spanning tree of a given graph in  $O(\log n)$  space.

The  $s$ - $t$  connectivity problem for *directed* graphs is NL-complete. This problem can be solved using  $O(\log^2 n)$  space and  $n^{O(\log n)}$  time by Savitch's algorithm (see [12]). Concerning polynomial-time algorithms solving this problem, the best known upper bound for space is the following slightly sublinear one due to Barnes et al. [6]:

**Theorem 2 (Barnes et al. [6]).** *Directed  $s$ - $t$  connectivity can be solved deterministically in  $n/2^{\Omega(\sqrt{\log n})}$  space and in polynomial time.*

This is also the best space upper bound for polynomial-time algorithms solving the following problems: computing the distance between a vertex  $s$  and a vertex  $t$  in an undirected or directed unweighted graph, computing the single-source shortest-path tree in a weighted undirected or directed graph [10], and a computing the breadth-first search tree.

## 2 Preliminaries

Throughout the paper, we assume that the set of vertices of a given graph is the set  $\{1, \dots, n\}$ .

We think of DFS in the following way: Initially, all the vertices are *white*. When vertex  $v$  is visited from vertex  $u$ , the color of  $v$  changes from white to *gray* and the *search head* moves *from  $u$  to  $v$* . When there is no more white neighbor of  $v$ , the search at  $v$  is finished, the color of  $v$  changes from gray to *black*, and the search head *returns from  $v$  to  $u$* .

Suppose that in a given undirected or directed graph,  $m$  vertices are reachable from the DFS starting vertex  $s$ . At time  $t = 0$ , all vertices are white. At time  $t = 1$ , the starting vertex  $s$  becomes gray. At each time  $t \geq 1$ , exactly one vertex changes its color, either from white to gray, or, from gray to black. At time  $t = 2m$ , the color of  $s$  becomes black and the search is completed. For a vertex  $v$ , the *discovery time* of  $v$  is the time when  $v$  changes its color from white to gray and the *finishing time* is the time when  $v$  changes its color from gray to black.

Note that the gray vertices always form a simple path from the starting vertex  $s$  to the vertex where the search head is currently located. We can also think of this path as residing in the *depth-first-search tree*.

We let  $\text{Reachable}(x, u, G)$  denote a subroutine that decides, given a graph  $G$  and two vertices  $x$  and  $u$ , whether vertex  $u$  is reachable from vertex  $x$  in  $G$ . If  $G$  is a directed graph, reachability is interpreted in terms of a directed path, and if  $G$  is undirected, it simply means connectivity. To implement  $\text{Reachable}(x, u, G)$  we apply Reingold’s algorithm and Barnes et al.’s algorithm for the cases of undirected and directed graphs, respectively.

### 3 Characterizations for the Gray and Black Vertices

In this section, for the sake of convenience, we collect our lemmas characterizing the gray vertices, the gray path, and the black vertices in several settings. These lemmas naturally yield our algorithms in the next section and they are crucial to explain their correctness. For most lemmas, proofs are immediate and omitted.

**Lemma 1 (All-White Path).** *Vertex  $v$  is visited during DFS while vertex  $u$  is gray, (i.e.,  $v$  is a descendant of  $u$  in the DFS tree) if and only if the following holds: At the time  $u$  is discovered,  $v$  is white and  $v$  can be reached from  $u$  by an all-white path.*

Let  $s$  be the starting vertex of a DFS. In the following we assume that the state of the DFS at time  $t$  is such that the search head is at a gray vertex  $u$ . Let  $p = \langle i_0 = s, i_1, \dots, i_{k-1}, i_k = u \rangle$  be the gray path at time  $t$ , where  $i_{j+1}$  is visited from  $i_j$  (for  $0 \leq j < k$ ).

The following lemma characterizes the gray path in terms of black vertices.

**Lemma 2 (Gray Path from Black Vertices).** *The gray path  $p = \langle i_0 = s, i_1, \dots, i_{k-1}, i_k = u \rangle$  satisfies the following. For  $j \in \{0, \dots, k-1\}$ , vertex  $i_{j+1}$  is the first vertex  $x$  in the adjacency list of vertex  $i_j$  such that (1)  $x$  is not black at time  $t$ , and that (2)  $x$  is not in  $\{i_0, \dots, i_j\}$ .*

**Proof.** Vertex  $i_{j+1}$  becomes gray only after all the vertices in the adjacency list of  $i_j$  preceding  $i_{j+1}$  have become non-white. □

Let  $C = \{i_0, \dots, i_k\}$  be the set of gray vertices comprising the gray path  $P$ . The following characterization explains how to reconstruct the path  $P$  from the set  $C$ .

**Lemma 3 (Gray Path from Gray Set).** *Let  $P' = \langle i_0, \dots, i_j \rangle$  be the initial segment of  $P$  of length  $j$ . Then, the following characterizes the immediate successor  $x = i_{j+1}$  of  $i_j$  in  $P$ .*

- (1)  $x$  is in  $C$ .
- (2)  $x$  is a neighbor of  $i_j$ .
- (3)  $x$  is not in  $\{i_0, \dots, i_j\}$ .
- (4)  $x$  is the first vertex in the adjacency list of  $i_j$  satisfying (1), (2), and (3).

For our algorithms we need to be able to reconstruct the gray path  $p = \langle i_0 = s, i_1, \dots, i_{k-1}, i_k = u \rangle$  from the two endpoints  $s$  and  $u$  alone. The following lemma characterizes the vertices  $i_1, \dots, i_{k-1}$  in such a way that one can reconstruct them given  $s$  and  $u$ . The proof immediately follows from Lemma 1 (All-White Path).

**Lemma 4 (Gray Path).** For  $j \in \{1, \dots, k-1\}$ , vertex  $i_j$  is the first vertex  $x$  in the adjacency list of  $i_{j-1}$  from which vertex  $u$  can be reached without going through any of the vertices in  $\{i_0, \dots, i_{j-1}\}$ .

**Corollary 1 (Gray Path Reconstruction).** Using an  $n$ -bit vector one can reconstruct  $i_1, \dots, i_{k-1}$  one by one as follows. For  $j = 1, \dots, k-1$ , for each vertex  $x$  adjacent to  $i_{j-1}$ , use  $\text{Reachable}(x, u, G - \{i_0, \dots, i_{j-1}\})$  and the previous lemma to determine whether  $x$  is  $i_j$ .

When considering DFS on a DAG the characterization and reconstruction of the gray path simplifies as follows.

**Lemma 5 (Gray Path in a DAG).** For  $j \in \{1, \dots, k-1\}$ , vertex  $i_j$  is the first vertex  $x$  in the adjacency list of  $i_{j-1}$  such that vertex  $u$  is reachable from  $x$ .

**Proof.** Let  $P$  be a directed simple path from  $x$  to  $u$ . Then no vertex  $y$  in  $\{i_0, \dots, i_{j-1}\}$  can appear on the path  $P$  since  $x$  is reachable from  $y$  and the graph is acyclic.  $\square$

**Corollary 2 (Gray Path Reconstruction in a DAG).** For a DAG, one can reconstruct  $i_1, \dots, i_{k-1}$  similarly as in the Gray Path Reconstruction Corollary above but without keeping track of  $i_0, \dots, i_{j-1}$  by replacing the call to the routine  $\text{Reachable}(x, u, G - \{i_0, \dots, i_{j-1}\})$  with  $\text{Reachable}(x, u, G)$ .

Let  $s$  be a starting vertex of a DFS. Assume that at time  $t-1$ , the gray path is of the form  $\langle i_0 = s, \dots, i_k = u, i_{k+1} = v \rangle$ , and that at time  $t$ , vertex  $v$  gets finished, and thus the gray path is now of the form  $\langle i_0 = s, \dots, i_k = u \rangle$ . Let the adjacency list of vertex  $u$  be  $\langle l_1, \dots, l_{q-1}, l_q = v, l_{q+1}, \dots, l_r \rangle$ . DFS has backtracked from  $v$  to  $u$  and now we want to find the first unvisited, white vertex  $x$  among  $l_{q+1}, \dots, l_r$  in order to visit  $x$  next. If we find out that such an  $x$  does not exist, we backtrack further from  $u$ .

Suppose  $y \in \{l_{q+1}, \dots, l_r\}$ . We can determine whether  $y$  has been visited or not, that is, whether  $y$  is black, gray or white using the following lemma.

**Lemma 6 (Black Vertex in a Directed Graph).** Vertex  $y$  is black at time  $t$  if and only if there exist  $j \in \{0, \dots, k\}$  and vertex  $\alpha$  such that the following hold:

1. The directed edge  $(i_j, \alpha)$  exists.
2. In the adjacency list of  $i_j$ , vertex  $\alpha$  precedes vertex  $i_{j+1}$ .
3. Vertex  $y$  is reachable from vertex  $\alpha$  without going through any of the vertices  $\{i_0, \dots, i_j\}$ .

**Proof.** Vertex  $y$  is black if and only if the path from  $s$  to  $y$  in the DFS tree is lexicographically smaller than the path from  $s$  to  $u$ . We can easily finish the proof using the Lemma 1 (All-White Path).  $\square$

For undirected graphs, we can simplify the lemma above as follows.

**Lemma 7 (White-Black Not Adjacent in Undirected DFS).** During a DFS in an undirected graph, a white vertex is never adjacent to a black vertex.

**Proof.** Initially, the property holds, and the property is maintained during DFS since a vertex becomes black only when all of its neighbors are non-white.  $\square$

**Lemma 8 (Black Vertex in an Undirected Graph).** *Vertex  $y$  is black at time  $t$  if and only if the following holds: There exists a vertex  $w \in \{l_1, \dots, l_{q-1}\}$  such that  $y$  is reachable from  $w$  without going through  $\{i_0, \dots, i_k\}$ .*

**Proof.** When vertex  $u$  is first visited from  $i_{k-1}$ , none neighbor of  $u$  is black by Lemma 7 (White-Black Not Adjacent in Undirected DFS), and hence  $j$  and  $\alpha$ , as required in Lemma 6 (Black Vertex in Directed Graph), cannot exist if  $j < k$ .  $\square$

## 4 $O(n)$ -Space DFS Algorithms

Our algorithms maintain the color of each vertex. For example, our 4-color algorithm uses 2 bits per vertex to hold the current color and thus uses  $2n$  bits in total for color information. Any additional space used is  $o(n)$ , and thus the space used to maintain the colors dominates the space complexity of our algorithms.

A basic problem that we face when restricted to  $O(n)$  space is that we cannot store, for example, the ordered list of the vertices that are currently gray since that would require  $\Theta(n \log n)$  space. A basic solution is to retrieve information by restarting the search from the starting vertex.

**Algorithm 1: a 4-Color Algorithm.** Our first algorithm, Algorithm 1, uses 4 colors for each vertex. It uses white, gray, and black according to the definitions of these colors explained in Section 2. To backtrack, it retraces the current gray path using Lemma 3 (Gray Path from Gray Set) by using one new color, blue, to keep track of the gray vertices in the initial segment reconstructed so far. We describe the algorithm in greater detail. Initially the starting vertex is colored gray and all other vertices are colored white. Suppose during the DFS we are at a vertex  $u$  (which must then be gray). If  $u$  has a white neighbor then we proceed with the search going to the first white neighbor of  $u$ , which is then colored gray. If  $u$  has no white neighbor, we color  $u$  black and backtrack. When backtracking, in case  $u$  is the starting vertex the search ends. Otherwise we need to determine the parent of  $u$  which is done by retracing the gray path as follows. We color the starting vertex blue. Then we repeatedly find the smallest gray vertex  $v$  that is a neighbor of the last vertex that was colored blue until this vertex is  $u$ . When  $u$  is colored blue, the parent of  $u$  is the last vertex that was colored blue just before that. We then recolor all blue vertices to be gray (by once again retracing the path), recolor  $u$  to be black, and have successfully backtracked.

**Theorem 3.** *Given an undirected or directed graph  $G$  consisting of  $n$  vertices and  $m$  edges, DFS on  $G$  can be done in  $O(mn)$  time and in  $2n + O(\log n)$  space.*

**Proof.** As explained in Section 2, the total number of color changes is  $2n$ . Between any two color changes, each edge is inspected at most  $O(1)$  times.  $\square$

**Algorithm 2: a Faster 4-Color Algorithm.** We can speed up Algorithm 1 while still using only  $O(n)$  bits as follows. Algorithm 2 uses a double-ended queue (DEQ) holding  $O(n/\log n)$  items, where each item in the queue is the  $O(\log n)$ -bit name of some vertex. The queue holds the most recently visited  $O(n/\log n)$  gray vertices.

Backtracking to vertex  $u$  can be done in  $O(1)$  time if the queue holds  $u$ , that is, if the queue is nonempty. When the queue becomes empty, we reconstruct the gray path in  $O(m)$  time, but such reconstructions happen at most  $O(\log n)$  times. Thus we have the following.

**Theorem 4.** *DFS on undirected and directed graphs can be done in  $O(m \log n)$  time and in  $O(n)$  space.*

**Remark 1:** When using space  $s$ , where  $n \leq s \leq n \log n$ , the algorithm above can be adjusted to run in time  $O(m n \log n/s)$ . Thus this algorithm is *memory-adjustable* in the sense of [3], [4], and [11].

**Algorithm 3: a 3-Color Algorithm.** By repeatedly restarting, we can reduce the number of colors by one. In each iteration, Algorithm 3 identifies *one* new black vertex. Starting from vertex  $s$ , we proceed using colors white, gray and black until we find the first gray vertex  $v$  that is now changing its color from gray to black. At such a point, we globally update the color of  $v$  as black, change the color of all the other gray vertices *from gray back to white*, and start a new iteration, again from  $s$ . Correctness of Algorithm 3 follows from Lemma 2 (Gray Path from Black Vertices).

**Remark 2:** In the description of Algorithm 3 above, vertex  $x$  being white does not always imply that  $x$  has never been visited: Even when  $x$  has been visited, the color of  $x$  becomes white again in a new iteration on the black-or-white graph.

With Algorithm 3 we have obtain following theorem.

**Theorem 5.** *For every  $\varepsilon > 0$ , DFS on undirected and directed graphs can be done in  $O(mn)$  time and in  $(\log_2(3) + \varepsilon + o(1))n$  space.*

**Three Situations of a DFS.** To describe our next two algorithms we think of the following three situations in which a DFS algorithm can be:

1. **First visit:** This situation arises if a vertex  $v$  has just been visited for the first time. The successor of  $v$  will be the first white vertex in the adjacency list of  $v$  if such a vertex exists. Otherwise we backtrack.
2. **Backtrack:** When vertex  $v$  becomes black, we backtrack to the parent vertex  $u$ .
3. **Pivot:** After backtracking from  $v$  to  $u$ , if the adjacency list of  $u$  is  $\langle l_1, \dots, l_q = v, l_{q+1}, \dots, l_r \rangle$ , we wish to find the first white vertex  $x$  among  $l_{q+1}, \dots, l_r$ , and visit  $x$  next if such an  $x$  exists; otherwise we backtrack further.

We now describe algorithms in terms of how they proceed in each of the three situations.

**Algorithm 4: a 2-Color Algorithm.** Our two color algorithm maintains the gray path but does not distinguish between black and white vertices. The three situations of the DFS are handled in the following way.

1. First visit: When  $v$  has just been visited for the first time, the color of  $v$  had been white and has just become gray. This situation is essentially a special case of the situation Pivot except that there is no vertex  $v$  from which we have just returned. To reduce this situation to the pivot situation, we simply pretend that there is an auxiliary black vertex  $v$  not part of the actual input graph incident only to the edge  $(u, v)$  from which we have just backtracked to  $u$ . For this, the vertex  $v$  is treated as the first neighbor of  $u$ .
2. Backtrack: In order to backtrack we memorize the current vertex  $u$  and then retrace the gray path from the starting vertex  $s$  to  $u$ . To do so, we first reinitialize so that all vertices are white. Using Corollary 1 (Gray Path Reconstruction), we iteratively reconstruct the gray path from the starting vertex  $s$  to  $u$ , and thereby find  $u$ 's parent. In applying Corollary 1 we use the Reachable routine and one color, gray.
3. Pivot: Using Corollary 1 (Gray Path Reconstruction) together with Lemmas 6 and 8 (Black Vertex in Directed/Undirected Graph), we find the white vertex  $x$  to be visited next by using the Reachable routine and using one color, gray. If no such vertex  $x$  exists, we perform a backtracking step.

The space used by Algorithm 4 is  $n$  bits plus the space used by the routine Reachable.

**Algorithm 5: an algorithm without colors for DAGs.** For the case of DAGs, we do not need to use any color. The algorithm copies the behavior of Algorithm 4, which in the case the input graph is a DAG simplifies as follows.

1. First visit: As in the case of Algorithm 4, this situation reduces to the pivot situation.
2. Backtrack: Similar to the Backtrack situation of Algorithm 4, we can just retrace the gray path from the starting vertex  $s$  to the current vertex  $u$ . Applying Corollary 2 instead of Corollary 1 we always use  $\text{Reachable}(x, u, G)$  instead of  $\text{Reachable}(x, u, G - \{i_0, \dots, i_{j-1}\})$ , thus avoid using any colors.
3. Pivot: Again Similar to the respective situation in Algorithm 4, we can follow the gray path by reconstructing one gray edge at a time, thereby forgetting the previous gray edges, and by then invoking  $\text{Reachable}(x, u, G)$ .

With Algorithms 4 and 5 and Theorems 1 and 2, we can conclude as follows.

**Theorem 6.** (1) DFS on a directed graph can be done in  $n + n/2^{\Omega(\sqrt{\log n})}$  space and in polynomial time.

(2) DFS on an undirected graph can be done in  $n + O(\log n)$  space and in polynomial time.

(3) DFS on a DAG can be done either in space  $n/2^{\Omega(\sqrt{\log n})}$  and in polynomial time or in space  $O(\log^2 n)$  and in time  $n^{O(\log n)}$ .

## 5 Tree-Walking

In this section, we consider undirected trees and forests. We give a simple  $O(\log n)$ -space algorithm for the depth-first traversal of a tree. The algorithm can be extended to an  $O(\log n)$ -space algorithm for deciding whether two given vertices are connected in a given forest and to an  $O(\log n)$ -space algorithm for deciding if a given undirected graph contains a cycle.

Throughout the section, we assume that each vertex  $u$  holds a cyclic list of its neighbors, and  $Next_u(v)$  denotes the vertex that immediately follows vertex  $v$  in the cyclic list of vertex  $u$ . We can follow a tree in the depth-first order starting from any edge  $(u, v)$  as follows:

```

procedure EdgeFollow( $u, v$ )
   $c = 0; (u_0, v_0) = (u, v);$ 
  repeat
    Output the edge  $(u, v);$ 
     $w = Next_v(u);$ 
     $u = v; v = w; c = c + 1;$ 
  until  $u = u_0$  and  $v = v_0;$ 
  return  $c;$ 

```

**Lemma 9.** *Let  $G$  be a tree of  $n$  vertices and  $(u, v)$  be any edge. Then, the procedure EdgeFollow( $u, v$ ) returns  $2n - 2$  after visiting every edge of  $G$  exactly twice.*

**Proof.** We prove the lemma by induction on  $n$ . Let  $Adj(u) = (v = v_0, v_1, \dots, v_a)$  and  $Adj(v) = (u = u_0, u_1, \dots, u_b)$  be the cyclic adjacency list of  $u$  and  $v$ , respectively. We also assume that removal of  $(u, v)$  from  $G$  results in two trees  $T_u$  and  $T_v$ , where  $T_u$  (resp.  $T_v$ ) is the tree containing the vertex  $u$  (resp.  $v$ ). By the induction hypothesis, if we apply the procedure EdgeFollow( $u, v$ ) to the tree  $(u, v) + T_v$ , then it returns  $2|T_v|$  after traversing all the edges in  $(u, v) + T_v$ . Similarly, applying the procedure EdgeFollow( $v, u$ ) to the tree  $(v, u) + T_u$ , it traverses all the edges of  $T_u$  and returns  $2|T_u|$ . Since  $Next_u(v) = v_1$  and  $Next_v(u_b) = u$ , we can combine the two edge sequences produced by EdgeFollow( $u, v$ ) and EdgeFollow( $v, u$ ) to obtain the complete sequence of the depth-first search on  $T$ . The lemma follows since  $|T_u \cup T_v| = n$  and the edge  $(u, v)$  is contained twice in each of the sequences. Note that the exceptional cases when  $T_u$  or  $T_v$  is empty are dealt with appropriately.  $\square$

**Theorem 7.** *Using Edge-Follow, deciding  $s$ - $t$  connectivity in a forest and detecting a cycle in an undirected graph can both be done in  $O(\log n)$  space and in  $O(n)$  time.*

## 6 DFS in $O(\log n)$ -Space for Undirected Graphs with $O(1)$ -Size Feedback Vertex Set

Now we consider the case where the input is an undirected graph  $G = (V, E)$  having feedback vertex set  $F$  of constant size. A *feedback vertex set*  $F \subseteq V$  of  $G$

is a set of vertices such that  $G - F$  contains no cycle. If a graph has a feedback vertex set of constant size, we can easily find a constant-size feedback vertex set in polynomial time and logarithmic space  $O(|F| \log n)$ . Thus, from now on, assume that we are given an undirected graph  $G = (V, E)$  and a feedback vertex set  $F \subseteq V$  of constant size. Furthermore, without loss of generality, assume that the set  $F$  includes a DFS-starting vertex  $s$ . (We may always add  $s$  to  $F$  if  $s \notin F$ .) Note that  $G - F$  is a forest.

Our overall strategy will be similar to Algorithm 4. By exploiting the fact that  $G - F$  is a forest, we can proceed similarly as in Algorithm 4 without remembering all the gray vertices and without using any color for reconstructing the gray path.

In particular, for the current gray path  $P$ , we only keep in memory the following parts of  $P$ : the vertices  $x$  in the feedback set  $F$  that appear in  $P$ , together with appearance order information, and for each such  $x$ , the non-feedback vertices that immediately precede and follow  $x$  in  $P$  if such vertices exist.

For simplicity of explanation, assume that precisely four vertices in the set  $F$ , namely vertices  $i, j, k$  and  $l$ , appear in the current gray path  $P$  in this order. Then, we will keep in memory the following parts of path  $P$  as our data (We do *not* keep in memory the parts corresponding to "..."):

$$i, i_1, \dots, j_0, j, j_1, \dots, k_0, k, k_1, \dots, l_0, l, l_1, \dots, l_c,$$

where all of the above are in the order of appearance in  $P$ ,  $i$  is the starting vertex  $s$ ,  $l_c$  is the vertex where the current head lies, and, for example,  $j_0, j$ , and  $j_1$  are consecutive vertices in the path  $P$ . It may happen that, for example,  $i_1 = j_0$ , or that neither  $i_1$  nor  $j_0$  exist (in this case two feedback vertices  $i$  and  $j$  appear consecutively in  $P$ ). Since the size of the feedback vertex set  $F$  is constant, the data above has size only  $O(\log n)$ .

How can we reconstruct the whole gray path  $P$  from the data above? Consider, for example, the vertices appearing between  $j_1$  and  $k_0$  in the path  $P$ . These vertices, together with the appearance ordering, can be characterized as vertices appearing in the unique simple path connecting vertices  $j_1$  and  $k_0$  in the forest  $G - F$ .

Note that we can follow the vertices appearing in path  $P$  consecutively, one at a time, using only the tree-walking algorithm and without using Reingold's  $s$ - $t$  connectivity algorithm. We can thus in space  $O(\log n)$  determine at any point in time whether a vertex is gray and within the same space complexity compute the predecessor of a vertex on the gray path.

In analogy to Algorithm 4 it remains to explain how to perform the pivot situation in a DFS. In order to proceed as in Algorithm 4, we have to be able to check whether vertices  $y$  and  $z$  are connected in  $G - C$ , where  $C$  is the set of the vertices that are currently gray and vertices  $y$  and  $z$  are two neighbors of the vertex  $u$  to which we have just backtracked.

We claim that connectivity checking for the restricted case of constant-size feedback vertex set can be done using only the tree-walking algorithm (again without using Reingold's  $s$ - $t$  connectivity algorithm) as follows. Since paths without internal vertices from the feedback vertex set and without gray vertices can



be found using the tree-walking algorithm, it suffices to show that for each pair of vertices in the feedback vertex set we can determine whether they are connected in  $G - C$ .

For each pair of feedback vertices  $f$  and  $g$ , we can determine, using tree-walking, whether  $f$  and  $g$  are connected by a path going only through non-feedback vertices that are not gray. So, by first checking whether  $f$  and  $g$  are connected in  $G - (C \cup F)$ , where  $C$  is the set of vertices that are currently gray, and taking the transitive closure, we can obtain the complete table as to which pairs of feedback vertices are connected in  $G - C$ . Note that since  $F$  has constant size, taking the transitive closure can be done in space  $O(\log n)$ . We conclude the following.

**Theorem 8.** *For undirected graphs whose minimum feedback vertex set are of constant size, DFS can be done in  $O(\log n)$  space using only the tree-walking algorithm (i.e., without appealing to Reingold's  $s$ - $t$  connectivity algorithm).*

## 7 Open Questions

We conclude with two open questions.

**Open Question 1:** Can DFS be done in  $cn$  space for some constant  $c < 1$ ?

**Open Question 2:** Using  $O(n)$  space, can DFS be done in  $o(m \log n)$  time?

## References

1. Aggarwal, A., Anderson, R.: A Random NC Algorithm for Depth-First Search. *Combinatorica* **8**(1), 1–12 (1988)
2. Anderson, R., Mayr, E.: Parallelism and the Maximal Path Problem. *Information Processing Letters* **24**(2), 121–126 (1987)
3. Asano, T., Elmasry, A., Katajainen, J.: Priority Queues and Sorting for Read-Only Data. In: Chan, T.-H.H., Lau, L.C., Trevisan, L. (eds.) TAMC 2013. LNCS, vol. 7876, pp. 32–41. Springer, Heidelberg (2013)
4. Asano, T., Kirkpatrick, D.: Time-Space Tradeoffs for All-Nearest-Larger-Neighbors Problems. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 61–72. Springer, Heidelberg (2013)
5. Asano, T., Kirkpatrick, D., Nakagawa, K., Watanabe, O.:  $\tilde{O}(\sqrt{n})$ -Space and Polynomial-time Algorithm for the Planar Directed Graph Reachability Problem. ECCO Report 71 (2014); also. In: Ésik, Z., Csuhaj-Varjú, E., Dietzfelbinger, M. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 45–56. Springer, Heidelberg (2014)
6. Barnes, G., Buss, J., Ruzzo, W., Schieber, B.: A Sublinear Space, Polynomial Time Algorithm for Directed  $s$ - $t$  Connectivity. *SIAM Journal of Computing* **27**(5), 1273–1282 (1998)
7. Elberfeld, M., Jakobý, A., Tantau, T.: Logspace Versions of the Theorems of Bodlaender and Courcelle. In: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010), pp. 143–152 (2010)
8. Elberfeld, M., Kawarabayashi, K.: Embedding and Canonizing Graphs of Bounded Genus in Logspace. In: Proceedings of the 46th Annual ACM Symposium on the Theory of Computing (STOC 2014), pp. 383–392 (2014)

9. Imai, T.: Polynomial-Time Memory Constrained Shortest Path Algorithms for Directed Graphs. In: Proceedings of the 12th Forum on Information Technology, vol. 1, pp. 9–16 (2013) (in Japanese)
10. Imai, T., Nakagawa, K., Pavan, A., Vinodchandran, N., Watanabe, O.: An  $O(n^{1/2+\epsilon})$ -Space and Polynomial-Time Algorithm for Directed Planar Reachability. In: Proceedings of 2013 IEEE Conference on Computational Complexity, pp. 277–286 (2013)
11. Konagaya, M., Asano, T.: Reporting All Segment Intersections Using an Arbitrary Sized Work Space. IEICE Transactions **96-A**(6), 1066–1071 (2013)
12. Papadimitriou, C.: Computational complexity. Addison-Wesley (1994)
13. Reif, J.: Depth-First Search Is Inherently Sequential. Information Processing Letters **20**(5), 229–234 (1985)
14. Reingold, O.: Undirected Connectivity in Log-Space. Journal of the ACM **55**(4), 17:1–17:24 (2008)
15. Tarjan, R.: Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972)
16. de la Tore, P., Kruskal, C.: Fast Parallel Algorithms for Lexicographic Search and Path-Algebra Problems. Journal of Algorithms **19**, 1–24 (1995)
17. de la Tore, P., Kruskal, C.: Polynomially Improved Efficiency for Fast Parallel Single-Source Lexicographic Depth-First Search, Breadth-First Search, and Topological-First Search. Theory of Computing Systems **34**, 275–298 (2001)

# Dynamic Path Counting and Reporting in Linear Space

Meng He<sup>1</sup>, J. Ian Munro<sup>2</sup>, and Gelin Zhou<sup>2</sup>(✉)

<sup>1</sup> Faculty of Computer Science, Dalhousie University, Halifax, Canada  
mhe@cs.dal.ca

<sup>2</sup> David R. Cheriton School of Computer Science, University of Waterloo,  
Waterloo, Canada  
{imunro,g5zhou}@uwaterloo.ca

**Abstract.** In the path reporting problem, we preprocess a tree on  $n$  nodes each of which is assigned a weight, such that given an arbitrary path and a weight range, we can report the nodes whose weights are within the range. We consider this problem in dynamic settings, and propose the first non-trivial linear-space solution that supports path reporting in  $O((\lg n / \lg \lg n)^2 + occ \lg n / \lg \lg n)$  time, where  $occ$  is the output size, and the insertion and deletion of a node of an arbitrary degree in  $O(\lg^{2+\epsilon} n)$  amortized time, for any constant  $\epsilon \in (0, 1)$ . Obvious solutions based on directly dynamizing solutions to the static version of this problem all require  $\Omega((\lg n / \lg \lg n)^2)$  time for each node reported, and thus our query time is much faster. For the counting version of this problem, we design a structure that supports path counting in  $O((\lg n / \lg \lg n)^2)$  time, and insertion and deletion in  $O((\lg n / \lg \lg n)^2)$  amortized time. This matches the current best result for 2D dynamic range counting, which can be viewed as a special case of path counting.

## 1 Introduction

In computer science, trees are widely used in modeling and representing different types of data. In many scenarios, objects are represented by nodes and their properties are characterized by weights assigned to nodes. Researchers have studied the problems of maintaining a weighted tree, such that, given any pair of nodes, certain functions over the path between these two nodes can be computed efficiently [1, 5–7, 14, 15, 17, 18, 21]. The inquiries of the values of these functions are referred to as *path queries*.

Previously, most work on path queries focus on static weighted trees, i.e., the structure and the weights of nodes remain unchanged over time. This assumption is not always realistic and it is highly inefficient to rebuild the whole data structure when handling updates. In this paper, we consider the problem of maintaining dynamic weighted trees and design data structures that support *path counting* and *path reporting* queries in linear space and efficient time. More

---

This work was supported by NSERC and the Canada Research Chairs Program.

precisely, given a query path and a query range, these types of queries return the number/set of nodes on the path whose weights are in the given range. As mentioned in He et al.'s work [14, 15], these path queries generalize two-dimensional range counting and reporting queries.

Without loss of generality, we represent the input tree as an ordinal one, i.e., a rooted tree in which children of a node are ordered. Our data structures allow to change the weight of an existing node, insert a new node, or delete an existing node. These updates are referred to as `modify_weight`, `node_insert`, and `node_delete`, respectively. For `node_insert` and `node_delete`, we adopt the same powerful updating protocol as Navarro and Sadakane [20], which allows us to insert or delete a leaf, a root, or an internal node. A newly inserted internal node will become the parent of consecutive children of an existing node, and a deleted root must have only zero or one child. The deletion of a non-root node is described in Section 2.2.

It is natural to identify nodes with their preorder ranks in static ordinal trees. However, preorder ranks of nodes can change over time in dynamic trees. Thus, in our dynamic data structures, nodes are identified by immutable identifiers of sizes  $O(\lg n)$  bits<sup>1</sup>. Unless otherwise specified, the underlying model of computation in this paper is the unit-cost word RAM model with word size  $w = \Omega(\lg n)$ .

**Previous Work.** The problems of supporting static path counting and path reporting queries have been heavily studied in recent years [6, 14, 15, 21]. Given an input tree on  $n$  nodes whose weights are drawn from  $[1..σ]$ , He et al. [15] designed succinct data structures to support path counting queries in  $O(\lg σ / \lg \lg n + 1)$  time, and path reporting queries in  $O((occ + 1)(\lg σ / \lg \lg n + 1))$  time, where  $occ$  is the size of output. Later, Chan et al. [6] achieved more time/space trade-offs for path reporting queries. They developed an  $O(n)$ -word structure with  $O(\lg^\epsilon n + occ \cdot \lg^\epsilon n)$  query time, where  $\epsilon$  is an arbitrary constant in  $(0, 1)$ ; an  $O(n \lg \lg n)$ -word structure with  $O(\lg \lg n + occ \cdot \lg \lg n)$  query time; and an  $O(n \lg^\epsilon n)$ -word structure with  $O(\lg \lg n + occ)$  query time.

There are other heavily studied path query problems such as path minimum queries, and we refer to Chan et al. [6] for a recent survey on the static version of this problem. The dynamic version of the path minimum problem has also been studied extensively. Brodal et al. [5] designed a linear space data structure that supports queries and changes to the weight of a node in  $O(\lg n / \lg \lg n)$  time, and handles insertions or deletions of a node with zero or one child in  $O(\lg n / \lg \lg n)$  amortized time. The query time is optimal under the cell probe model provided that the update time is  $O(\lg^{O(1)} n)$  [2]. For the more restricted case in which only insertions and deletions of leaves are allowed, queries can be answered in  $O(1)$  time and updates can be supported in  $O(1)$  amortized time [1, 5, 17].

**Our Contributions.** We develop efficient dynamic data structures for path counting and path reporting queries, all of which occupy  $O(n)$  words. Our data structure supports path counting queries in  $O((\lg n / \lg \lg n)^2)$  time, and handles changes of weights, insertions and deletions in  $O((\lg n / \lg \lg n)^2)$  amortized time. This structure matches the best known result for dynamic range counting [12].

<sup>1</sup> We use  $\lg$  to denote the base-2 logarithm.

For path reporting queries, our data structure requires  $O(\lg^{2+\epsilon} n)$  time for updates, but answers queries in  $O((\lg n / \lg \lg n)^2 + occ \lg n / \lg \lg n)$  time, where  $occ$  is the output size. By slightly sacrificing the update time, our structure significantly improves the query time over the straightforward approaches that dynamize known static data structures [6, 14, 15, 21]: One could dynamize the static structure of He et al. [14] by replacing static labeled ordinal trees with dynamic unlabeled trees and dynamic bit vectors, and managing weight ranges using a red-black tree [8]. This leads to an  $O(n)$ -word data structure with  $O((1 + occ) \lg^2 n / \lg \lg n)$  query time and  $O(\lg^2 n / \lg \lg n)$  update time. Alternatively, one could obtain another  $O(n)$ -word structure with  $O(\lg^{2+\epsilon} n + occ \cdot (\lg n / \lg \lg n)^2)$  query time and  $O((\lg n / \lg \lg n)^2)$  update time, by dynamizing the improved result of He et al. [15] in a similar manner. It is unclear how to dynamize the structures designed by Patil et al. [21] and Chan et al. [6] within linear space.

All of our dynamic structures presented in this paper are able to handle insertions and deletions of nodes with multiple children, which are not supported in previous dynamic data structures for path queries [1, 5, 17]. Our approach is almost completely different from He et al.'s [15] approach for static path queries. To develop our data structures, we employ a various of techniques including *topology trees*, *tree extraction*, and *balanced parentheses*. In particular, for dynamic path reporting, one key strategy is to carefully design transformations on trees that preserve certain properties, such that the idea of dynamic fractional cascading can be adapted to work on multiple datasets in which each set represents tree-structured data. This new approach may be of general interest.

Section 2 reviews the techniques used in our data structures. Section 3 describes our dynamic data structures for path reporting. Due to the page limitation, the support for path counting is deferred to the full version of this paper.

## 2 Preliminary

### 2.1 Restricted Multilevel Partition and Topology Trees

Frederickson [9–11] proposed topology trees to maintain connectivity information and minimum spanning trees of dynamic graphs, and to support operations over dynamic trees. We make use of a variant of topology trees based on a restricted partition of a binary tree  $\mathcal{B}$ , where the nodes of  $\mathcal{B}$  are clustered into disjoint sets such that the elements in each set are nodes of a connected component of  $\mathcal{B}$ . Each of such components is called a *cluster*, and its *external degree* is the number of edges with exactly one endpoint being a vertex in the cluster. A *restricted partition of order  $s$*  of  $\mathcal{B}$  is defined to be a partition that satisfies the following conditions: each cluster has external degree at most 3; each cluster with external degree 3 contains only one node; each cluster with external degree less than 3 has at most  $s$  nodes, and no two adjacent clusters can be combined without breaking the above conditions. Frederickson gave a linear-time algorithm that creates a restricted partition of order  $s$  for a given binary tree on  $n$  nodes, and proved that the number of clusters is  $\Theta(\lceil n/s \rceil)$ .

The endpoints of the edges that connect different clusters are called *boundary nodes*. We further follow the notation of He et al. [13] and define the *preorder segments* of a cluster to be the maximal contiguous subsequences of nodes in the preorder sequence that are in the same cluster. Thus Frederickson's approach guarantees that each cluster has up to two boundary nodes and up to three preorder segments. The clusters including the root may have 3 preorder segments.

Frederickson further defined a *restricted multi-level partition* of a binary tree  $\mathcal{B}$  consisting of a set of  $h$  partitions of the nodes which can be computed recursively as follows: The clusters at level 0, which are called *base clusters*, are obtained by computing a restricted partition of order  $s$  of  $\mathcal{B}$ . Then, to compute the level- $l$  clusters for any level  $l > 0$ , we view each cluster at level  $l - 1$  as a node, and then compute a restricted partition of order 2 of the resulting tree. This recursion stops when the partition contains only one cluster containing all the nodes, which is the level- $h$  cluster.

A *topology tree*  $\mathcal{H}$  is defined for a restricted multi-level partition of a binary tree  $\mathcal{B}$ .  $\mathcal{H}$  contains  $h + 1$  levels. Each node of  $\mathcal{H}$  at level  $l$  represents a level- $l$  cluster, and the up to two children of a node at level  $l$  each corresponds to one of the two level- $(l - 1)$  clusters that this level- $l$  cluster consists of. Additional links are maintained between each pair of adjacent nodes at the same level of  $\mathcal{H}$ . Frederickson proved that  $h = O(\lg n)$ . Topology trees were used in maintaining a dynamic forest of binary trees, to support two operations: **link** which combines two trees in the forest into one by adding an edge between the root of one binary tree and an arbitrary given node of the other that has less than two children, and **cut** which breaks one tree into two by removing an arbitrary given edge. The following lemma summarizes a special case of their results to be used in our solutions, in which we say that a cluster is *modified* during updates if it is deleted or created during this update, its nodes or edges have been changed or an edge with an endpoint in the cluster has been inserted or deleted:

**Lemma 1** ([10, 11]). *The topology trees of the binary trees in a given forest  $F$  on  $n$  nodes can be maintained in  $O(s + \lg n)$  time for each **link** and **cut**, where  $s$  is the maximum size of base clusters. Furthermore, each **link** or **cut** modifies  $O(1)$  clusters at any level of the topology trees maintained for the two binary trees updated by this operation, and once a cluster is modified, the clusters represented by the ancestors of its corresponding node in the topology tree are all modified. These topology trees have  $\Theta(f + n/s)$  nodes in total, where  $f$  is the current number of trees in  $F$ , and occupy  $O(S + (f + n/s)\lg n)$  bits in total, where  $S$  is the total space required to store the tree structures of base clusters.*

## 2.2 Tree Extraction

Tree extraction has proved to be a powerful technique in supporting various types of static path queries [6, 14–16]. This technique is based on the deletion operation defined in the context of tree edit distance [4]. To delete a non-root node  $u$ , which is a child of  $v$ , the children of  $u$  are inserted in place of  $u$  in the

list of children of  $v$ , preserving the original order. Let  $T$  be an ordinal weighted tree and  $I$  be a weight range. For the sake of convenience, we add a dummy node  $r$  to be the new root of  $T$ , which has a NULL weight and will be the parent of the original root. We define  $T_I$  to be the extracted tree obtained by deleting all the non-root nodes whose weights are not in  $I$  from the argumentation of  $T$ . That is,  $T_I$  only consists of the dummy root and the nodes whose weights are in  $I$ . The crucial observation is that tree extraction preserve the ancestor-descendant, preorder, and postorder relationships among the remaining nodes.

### 3 Dynamic Path Reporting

Let  $T$  be a dynamic tree on  $n$  weighted nodes. W.l.o.g, we assume that node weights are distinct. We construct a weight-balanced B-tree [3],  $W$ , with leaf parameter 1 and branching factor  $d = \lceil \lceil \lg n \rceil^\epsilon \rceil$  for any positive constant  $\epsilon$  less than  $1/2$ . When the value of  $d$  changes due to updates, we reconstruct the entire data structure and amortize the cost of rebuilding to updates. By the properties of weight-balanced B-trees, each internal node of  $W$  has at least  $d/4$  and at most  $4d$  children, and the only exception is the root which is allowed to have fewer children. Each leaf of  $W$  represents a weight range  $[a, b)$ , where  $a$  and  $b$  are weights assigned to nodes of  $T$ , and there is no node of  $T$  whose weight is between  $a$  and  $b$ . An internal node of  $W$  represents a (contiguous) range which is the union of the ranges represented by its children, where the children are sorted by the left endpoints of these weight ranges. The levels of  $W$  are numbered  $0, 1, 2, \dots, t$ , starting from the leaf level, where  $t = O(\lg n / \lg \lg n)$  denotes the number of the root level. The tree structure of  $W$  together with the weight range represented by each node is maintained explicitly.

For each internal node  $v$  of  $W$ , we conceptually construct a tree  $T(v)$  as follows: Let  $[a, b)$  denote the weight range represented by  $v$ . We construct a tree  $T_{[a,b)}$  consisting of nodes of  $T$  whose weights are in  $[a, b)$  using the tree extraction approach described in Section 2.2. For each node  $x$  in  $T_{[a,b)}$ , we then assign an integer label  $i \in [1..4d]$  if the weight of  $x$  is within the weight range of the  $i$ th child of  $v$ . The resulting labeled tree is  $T(v)$ .

We do not store each  $T(v)$  explicitly. Instead, we transform the tree structure of each  $T(v)$  into a binary tree  $\mathcal{B}(v)$  as in [6]: For each node  $x$  of  $T(v)$  with  $k > 2$  children denoted as  $x_1, x_2, \dots, x_k$ , we add  $k - 1$  dummy nodes  $y_1, y_2, \dots, y_{k-1}$ . Then,  $x_1$  and  $y_1$  become the left and the right children of  $x$ , respectively. For  $i = 1, 2, \dots, k - 2$ , the left and the right children of  $y_i$  are set to be  $x_{i+1}$  and  $y_{i+1}$ , respectively. Finally,  $x_k$  becomes the left and only child of  $y_{k-1}$ . In  $\mathcal{B}(v)$ , the node corresponding to the dummy root of  $T(v)$  is also considered a dummy node, and a node is called an *original node* if it is not a dummy node. We observe that this transformation preserves the preorder and postorder relationships among the original nodes in  $T(v)$ . Furthermore, the set of original nodes along the path between any two original nodes remains unchanged after transformation. Each original node in  $\mathcal{B}(v)$  is associated with its label in  $T(v)$ , which is an integer in  $[1..4d]$ , while each dummy node is assigned with label 0.

Let  $F_i$  denote the forest containing all the binary trees created for the nodes at the  $i$ th level of  $W$  for  $i > 0$ , i.e.,  $F_i = \{\mathcal{B}(v) : v \text{ is a node at the } i\text{th level of } W\}$  for  $i \in [1..t]$ . Thus  $F_t$  contains only one binary tree which corresponds to the root of  $W$ , and this tree contains all the nodes of the given tree  $T$  as original nodes. This allows us to maintain a bidirectional pointer between each node in  $T$  and its corresponding original node in  $F_t$ .

$W$  and  $T$  are stored using standard, pointer-based representations of trees. In the rest of this section, we first present, in Section 3.1, a data structure that can be used to maintain a dynamic forest in which each node is assigned a label from an alphabet of sub-logarithmic size, to support a set of operations including *path summary* queries which is to be defined later. This structure is of independent interest and will be used to encode each  $F_i$ . We next show, in Section 3.2, how to maintain pointers between forests constructed for different levels of  $W$ , which will be used to locate appropriate nodes of these forests when answering path reporting queries. Finally we describe how to answer path reporting queries and perform updates in weighted trees in Section 3.3.

### 3.1 Representing Dynamic Forests with Small Labels to Support Path Summary Queries

We now describe a data structure which will be used to encode  $F_i$  in subsequent subsections. As this structure may be of independent interest, we formally describe the problem its addresses as follows. Let  $F$  be a dynamic forest of binary trees on  $n$  nodes in total, in which each node is associated with a label from the alphabet  $[0..\sigma]$ , where  $\sigma = O(\lg^\epsilon n)$  for an arbitrary constant  $\epsilon \in (0, 1/2)$ . Our objective is to maintain  $F$  to support **link**, **cut** and the following operations:

- **parent** $_\alpha(x)$ : return the  $\alpha$ -parent of node  $x$ , i.e., the lowest ancestor of  $x$  that has label  $\alpha$ , which can be  $x$  itself.
- **LCA** $(x, y)$ : return the lowest common ancestor of two given nodes  $x$  and  $y$  residing in the same binary tree.
- **pre\_succ** $_\alpha(x)$ : return the  $\alpha$ -successor of  $x$  in preorder, i.e., the first  $\alpha$ -node in preorder that succeeds  $x$  (this could be  $x$  itself).
- **post\_pred** $_\alpha(x)$ : return the  $\alpha$ -predecessor of  $x$  in postorder, i.e., the last  $\alpha$ -node in postorder that precedes  $x$  (this could be  $x$  itself).
- **summary** $(x, y)$ : given two nodes  $x$  and  $y$  residing in the same binary tree, return a bit vector of  $\sigma + 1$  bits in which the  $\alpha$ th bit is 1 iff there exists an  $\alpha$ -node along the path from  $x$  to  $y$ . This query is called path summary.
- **modify** $(x, \alpha)$ : change the value of  $x$ 's label to  $\alpha$ .

We first set  $s = \lceil \frac{\lceil \lg n \rceil}{\lg \lceil \lg n \rceil} \rceil$  in Lemma 1, and use the lemma to maintain the topology trees of the binary trees in  $F$ . We call each base cluster a *micro-tree*.

We next define a subset of levels of the topology trees *marked levels*. For  $i = 0, 1, \dots$ , the  $i$ th marked level of a topology tree is level  $i \lceil \epsilon \lg \lg n \rceil$  of this topology tree. Since in a topology tree, the restricted partition at each level except level 0 is of order 2, each internal node of the topology tree has at most



two children. Therefore, for  $i \geq 1$ , each cluster at the  $i$ th marked level contains at most  $2^{\lceil \epsilon \lg \lg n \rceil} \leq 2^{\epsilon \lg \lg n} = \lg^\epsilon n$  clusters at the  $(i - 1)$ st marked level. We then define the *macro-tree* for a node at the  $i$ th marked level of a topology tree, for  $i \geq 1$ , to be the tree obtained by viewing each cluster at the  $(i - 1)$ st marked level as a single node and adding an edge between two of these nodes if and only if their corresponding clusters are adjacent. As shown in the discussion above, each macro-tree is a binary tree with at most  $\lg^\epsilon n$  nodes. A macro-tree is called a *tier- $i$  macro-tree* if it is constructed for a node at the  $i$ th marked level. A node in a tier- $i$  macro-tree is called a *boundary node* if its corresponding cluster contains the endpoint of an edge that has only one endpoint in this tier- $i$  macro-tree. By the properties of restricted multi-level partitions, each macro-tree has at most two boundary nodes and at most one of them is a leaf in the macro tree.

We construct auxiliary data structures for each micro-tree and macro-tree. Our main idea is to create structures that can fit in  $\frac{1}{2} \lg n$  bits (in addition to maintaining pointers such as those that can be used to map macro tree nodes to macro trees at the lower marked level), so that we can construct  $o(n)$ -bit lookup tables to perform operations in each micro-tree or macro tree. Operations over  $F$  are then supported by operating on a constant number of micro-trees and a constant number of macro-trees at each marked level. The proof of the following lemma is omitted due to the page limitation.

**Lemma 2.** *Let  $F$  be a dynamic forest of binary trees on  $n$  nodes in total, in which each node is associated with a label from the alphabet  $[0..\sigma]$ , where  $\sigma = O(\lg^\epsilon n)$  for any constant  $\epsilon \in (0, 1/2)$ .  $F$  can be represented in  $O(n \lg \lg n + f \lg n)$  bits to support `parent $_\alpha$` , `LCA`, `summary`, `pre_succ $_\alpha$` , `post_pred $_\alpha$`  and `modify` in  $O(\lg n / \lg \lg n)$  time, and `link` and `cut` in  $O(\lg^{1+\epsilon} n)$  time, where  $f$  is the current number of trees in  $F$ .*

### 3.2 Navigation Between Different Levels of $W$

As discussed previously, we use Lemma 2 to encode each  $F_i$  for  $i > 0$ . For each node at the  $i$ th level of  $W$ , we store a pointer to the root of its corresponding topology tree in  $F_i$ . Each tree node in  $F_i$  can be uniquely identified by a pointer to the micro-tree containing the node and its preorder rank in the micro-tree. We call this pair of pointer and preorder rank the *local id* of this node in  $F_i$ .

Since each node,  $x$ , of  $T$  appears once in  $F_i$  as an original node for each  $i \in [0..t]$ ,  $x$  has one local id at each level of  $W$ . In our algorithm for path reporting, given the local id of  $x$  in  $F_i$ , we need find its local id in  $F_{i-1}$  and  $F_{i+1}$ . Explicitly storing the answers would require too much space. Thus, our overall strategy is to precompute, for only a subset of nodes of  $T$ , their local ids in  $F_{i-1}$  and  $F_{i+1}$ . Then, we design an algorithm to compute local ids of other nodes, by making use of the fact that both tree extraction and our way of transforming each  $T(v)$  to  $\mathcal{B}(v)$  preserve relative preorder among nodes of  $T$ .

We now describe our strategy in details. In  $F_i$ , we call the clusters at the first marked level of the topology trees *mini-trees*. By our discussions in Section 3.1, each mini-tree then contains at most  $\lg^\epsilon n$  micro-trees, and has  $O(\lg^{1+\epsilon} n)$  tree

nodes. There is a one-to-one correspondence between mini-trees and tier-1 macro-trees, but they are conceptually different: each node in a mini-tree is a node of  $F_i$ , while a node in a tier-1 macro-tree represents a micro-tree. Because of this one-to-one correspondence, however, we do not distinguish the pointers to a tier-1 macro-tree from a pointer to its corresponding mini-tree. We say a micro-tree is the  $i$ th micro-tree of a mini-tree (or its corresponding tier-1 macro-tree), if this micro-tree is represented by the  $i$ th node in preorder of the tier-1 macro-tree corresponding to this mini-tree.

A node in  $F_i$  can also be uniquely identified by a pointer to the mini-tree containing the node and its preorder rank in the mini-tree. Conversions between this type of identification and the local id of the node can be done in constant time (the details are omitted). Thus we consider each of these two different identifiers as a valid local id of a node in  $F_i$  in the rest of the paper. Furthermore, we consider the support of  $\text{parent}_\alpha$  within any given mini-tree, i.e., given a node  $x$ , we are interested in finding its  $\alpha$ -parent in the same mini-tree if it exists. We also consider the following two operators over a mini-tree:

- $\text{pre\_rank}_\alpha(x)$ , which computes the number of  $\alpha$ -nodes preceding  $x$  in preorder (including  $x$  itself if it is labeled  $\alpha$ );
- $\text{pre\_select}_\alpha(i)$ , which locates the  $i$ th  $\alpha$ -node in preorder.

In the above definition, we allow  $\alpha$  be set to  $\bar{0}$ , which matches any label that is not 0. We have the following lemma. The proof is omitted here.

**Lemma 3.** *With  $o(n)$  additional bits,  $\text{parent}_\alpha$ ,  $\text{pre\_rank}_\alpha$  and  $\text{pre\_select}_\alpha$  can be supported in  $O(1)$  time over each mini-tree in  $F_i$ .*

We next define a set of pointers between mini-trees at different levels of  $W$  and we call these pointers *inter-level pointers*. These pointers are defined for each mini-tree  $\mu$  in any  $F_i$ . Let  $v$  be the node of  $W$  such that  $\mathcal{B}(v)$  contains  $\mu$ . If  $i < t$ , then for each preorder segment of  $\mu$ , we create an *up pointer* for the first original node,  $x$ , of this segment in preorder. This pointer points from  $x$  to the original node in  $F_{i+1}$  that corresponds to the same node of  $T$ . Next, if  $i > 1$ , for each preorder segment of  $\mu$  and for each label  $\alpha \in [1..4d]$ , if node  $y$  is the first node in this segment in preorder that is labeled  $\alpha$ , we store a *down pointer* from  $y$  to the original node in  $F_{i-1}$  that corresponds to the same node of  $T$  that  $y$  represents. No pointers are created for nodes labeled 0, as they are dummy nodes. So far we have created at most  $3(4d+1) = O(\lg^\epsilon n)$  inter-level pointers for each mini-tree, as each mini-tree has at most three preorder segments. Finally, we create a back pointer for each up or down pointer, doubling the total number of inter-level pointers created over all the levels of  $W$ .

To store inter-level pointers physically, we maintain all the pointers that leave from mini-tree  $\mu$  (again, suppose that  $\mu$  is in  $\mathcal{B}(v)$  which is part of  $F_i$ ) in a structure called  $P_\mu$ , including up and down pointers created for nodes in  $\mu$ , and back pointers for some of the up and down pointers created for mini-trees at adjacent levels of  $W$ . We further categorize these pointers into at most  $4d+1$  types: A type-0 pointer arrives at a mini-tree in  $F_{i+1}$ , i.e., goes to the level above,

and a type- $\alpha$  pointer for  $\alpha > 1$  arrives at a mini-tree in  $B(u)$ , where  $u$  is the  $\alpha$ th child of  $v$ . Note that it is possible that an up or down pointer of  $\mu$  and a back pointer from an adjacent level stored in  $P_\mu$  have the same source (a node in  $\mu$ ) and destination (a node in the forest for an adjacent level of  $W$ ). In this case, the back pointer is not stored separately in  $P_\mu$ , and hence each inter-level pointer in  $P_\mu$  can be uniquely identified by its type and the preorder rank of its source node in  $\mu$ . We also maintain the preorder rank of the first node of each of the (at most three) preorder segments in  $\mu$ . We use the approach of Navarro and Nekrich [19, Section A.3] with trivial modifications to encode  $P_\mu$  in  $|P_\mu|$  words, so that given a node  $x$  in  $\mu$ , we can retrieve in  $O(1)$  time the closest preceding node (this can be  $x$  itself) in the preorder segment of  $\mu$  containing  $x$  that has an inter-level pointer of a given type  $\alpha$ , as well as the local id of the destination of this pointer. Insertion and deletion of inter-level pointers can also supported in  $O(1)$  time. The details are deferred to the full version due to space limitation. We can now prove the following lemma:

**Lemma 4.** *Give the local id of a original node  $x$  in  $F_i$ , the local id of the original node in  $F_{i+1}$  (if  $i < t$ ) or  $F_{i-1}$  (if  $i > 1$ ) that represents the same node of  $T$  can be computed in  $O(1)$  time.*

*Proof.* We first show how to locate the node,  $y$ , in  $F_{i+1}$  that represents the same node of  $T$ . We start to find the closest node,  $x'$ , that precedes  $x$  in preorder, has a type-0 inter-level pointer, and resides in the same preorder segment,  $s_0$ , of the mini-tree containing  $x$  ( $x'$  is allowed to be  $x$  itself). The destinate node,  $y'$ , of this pointer is also retrieved during the same process, and it is a node in  $F_{i+1}$ . Node  $x'$  always exists because the first original node of each preorder segment in a mini-tree has an up pointer.

If  $x'$  happens to be  $x$  itself, then  $y'$  is  $y$  which is the answer. If not, we observe that  $y$  and  $y'$  are in the same preorder segment of a mini-tree in  $F_{i+1}$ . Suppose that  $u$  is the  $\alpha$ th node of  $v$ . It then follows that the number of  $\alpha$ -nodes of  $\mathcal{B}(v)$  that are between  $y'$  and  $y$  in preorder is equal to the number,  $k$ , of original nodes between  $x'$  and  $x$  in  $\mathcal{B}(u)$ . As the number of original and dummy nodes between  $x'$  and  $x$  in  $\mathcal{B}(u)$  is equal to the difference between the preorder ranks of  $x'$  and  $x$ , it suffices to compute the number of dummy nodes between them, which can be computed as  $\text{pre\_rank}_0(x) - \text{pre\_rank}_0(x')$  in  $\mathcal{B}(u)$ . By Lemma 3, this requires constant time since they are in the same mini-tree. Then, the preorder of  $y$  can be computed as  $\text{pre\_select}_\alpha(\text{pre\_rank}_\alpha(y') + k)$  in  $\mathcal{B}(v)$ , which again requires constant time. This gives us the local id of  $y'$ , and the entire process uses  $O(1)$  time. The node,  $z$ , in  $F_{i-1}$  that represents the same node of  $T$  as  $x$  does can be located using a similar process. □

### 3.3 Supporting Path Reporting

**Lemma 5.** *The structures in this section can answer a path reporting query in  $O((\lg n / \lg \lg n)^2 + \text{occ} \lg n / \lg \lg n)$  time, where  $\text{occ}$  is the output size.*

*Proof.* Let  $x$  and  $y$  be the two nodes that define the query path, and let  $[p, q]$  be the query weight range. We perform a top-down traversal in  $W$  to locate its up to two leaves that represent ranges containing  $p$  and  $q$ . During this traversal, at each level,  $i$ , of  $W$ , we visit at most two nodes of  $W$ , and each node to visit at the next level can be located using a binary search in  $O(\lg d) = O(\lg \lg n)$  time, as each node has at most  $4d$  children. As there are  $O(\lg n / \lg \lg n)$  levels in  $W$ , the total time required to determine the nodes of  $W$  to visit is  $O(\lg n)$ .

For each node,  $v$ , of  $W$  visited during the above top-down traversal, we also determine the original nodes  $x_v$  and  $y_v$  in  $\mathcal{B}(v)$  respectively corresponding to the lowest ancestors of  $x$  and  $y$  in  $T$  that are represented by nodes in  $\mathcal{B}(v)$  (each node is considered to be its own ancestor). These nodes are located during the top-down traversal as follows. Let  $u$  denote the parent of  $v$  in  $W$ , and suppose that  $v$  is the  $\alpha$ th child of  $u$ . Then to compute  $x_v$ , if  $x_u$  is labeled with  $\alpha$ , then we use Lemma 4 to locate  $x_v$  in constant time. Otherwise, we first locate  $x_u$ 's lowest  $\alpha$ -parent,  $x'$ , using Lemma 2 in  $O(\lg n / \lg \lg n)$  time, and then compute  $x_v$  as the node corresponding to  $x'$  in  $\mathcal{B}(v)$  in  $O(1)$  time using Lemma 4.  $y_v$  can be computed in a similar manner. The total time required to locate all these nodes in our query algorithm is thus  $O((\lg n / \lg \lg n)^2)$ .

For each node,  $v$ , of  $W$  visited during the traversal, if the range of at least one of  $v$ 's children is contained entirely in  $[p, q]$ , then we compute  $z_v = \text{LCA}(x_v, y_v)$  in  $\mathcal{B}(v)$ . We also perform a path summary query using  $x_v$  and  $y_v$  as the endpoints of the query path, and let  $V$  be the bit vector returned by the query. Suppose that the children of  $v$  whose ranges are contained in  $[p, q]$  are numbered  $j, j+1, \dots, k$ . Since  $V$  has  $O(\lg^\epsilon n)$  bits, then we can use an  $o(n)$ -bit table to retrieve the position of each 1 bit in  $V[j..k]$  in constant time. Then for each  $l \in [j..k]$  such that  $V[l] = 1$ , we claim that there are nodes along the path between  $x_v$  and  $y_v$  in  $\mathcal{B}(v)$  that are labeled  $l$ , and these nodes correspond to nodes of  $T$  to be reported. Each node from  $x_v$  to  $z_v$  (including  $x_v$  and  $z_v$ ) labeled  $l$  can be located using  $\text{parent}_l$  over  $\mathcal{B}(v)$  (when we reach a node whose preorder in  $\mathcal{B}(v)$  is less than or equal to that of  $z_v$ , we have located all these nodes), and for each node found, we keep finding its local id in the level above, until we find its local id at the root level of  $W$  which immediately gives us a node in  $T$ , and we report this node of  $T$ . The nodes from  $y_v$  to  $z_v$  (including  $x_v$  but excluding  $z_v$ ) labeled  $l$  can be located and have their corresponding nodes in  $T$  reported using the same approach. We observe that a constant number of  $\text{LCA}$  and  $\text{summary}$  are performed at each level of  $W$ , which require  $O((\lg n / \lg \lg n)^2)$  time in total. Then, for each node reported, only  $O(\lg n / \lg \lg n)$  time is spent: if we always charge each  $\text{parent}_\alpha$  operation to the last node reported before this operation is performed, then each node is charged a constant number of times, and the process described above which finds the node of  $T$  given its local id in  $\mathcal{B}(v)$  requires  $O(\lg n / \lg \lg n)$  time. This completes the proof.  $\square$

**Lemma 6.** *The structures in this section support `node_insert`, `node_delete` and `modify_weight` in  $O(\lg^{2+\epsilon} n)$  amortized time.*

*Proof.* We only show how to support `node_insert`; the other update operations can be handled similarly. Note that update operations may eventually change

the value of  $\lceil \lg n \rceil$ , but this can be handled by standard techniques of dynamic data structures.

Suppose that we insert a new node  $h$  with weight  $w_h$ . The new node  $h$  is inserted as a child of  $x$ , and a set of consecutive children of  $x$  between and including child nodes  $y$  and  $z$  become the children of  $h$  after the insertion. Here we consider the general case in which  $y$  and  $z$  exist and are different nodes; degenerate cases can be handled similarly. In the first step of our insertion algorithm, we insert the weight  $w_h$  into  $W$  by creating a new leaf for it. This may potentially cause the parent of this new leaf to split, but for now, we consider the case in which a split will not happen. The support for node splits in  $W$  is omitted due to the page limitation.

We next perform a top-down traversal of  $W$  to fix the structures created for the forest  $F_i$  at each level  $i$  of  $W$ . In our description, when we say node  $h$  (or  $x$ , etc.) in  $F_i$ , we are referring to the original node, either to be inserted to  $F_i$  or already exist in  $F_i$ , that corresponds to this node in  $T$ . At the top level, i.e., the  $t$ th level, of  $W$ , the forest  $F_t$  contains one single binary tree. As we maintain bidirectional pointers between nodes in  $T$  and nodes in  $F_t$ , we can immediately locate the nodes  $x$ ,  $y$  and  $z$  in  $F_t$ . Let  $x_1, x_2, \dots$  be the dummy nodes created for  $x$  in  $F_t$ , among which  $x_j$  and  $x_k$  are the dummy nodes that are parents of  $y$  and  $z$ , respectively. We then perform a constant number of updates to  $F_t$  as follows. First we perform the `cut` operation twice to remove the edge between  $x_{j-1}$  and  $x_j$ , and the edge between  $x_k$  and  $x_{k+1}$ . This divides  $F_t$  into three trees. We then create a tree on a new node  $x'_j$  which is a dummy node, and temporarily include this tree into  $F_t$ . Note that creating the topology tree and associated auxiliary data structures for a tree on a single node can be trivially done in constant time. We then replace the dummy node  $x_j$  by the node  $h$  to be inserted. This can be done by first performing binary searches in the ranges of the children of the root,  $r$ , of  $W$ , so that we know the correct label,  $\alpha$ , to assign to  $h$ . We then simply call `modify` to change the label, 0, assigned to  $x'_j$ , to  $\alpha$  using `modify`. We then perform `link` to add three edges so that  $x'_j$  becomes the right child of  $x_{j-1}$ ,  $h$  becomes the left child of  $x'_j$ , and  $x_{k+1}$  becomes the right child of  $x'_j$ . It is clear that all these operations require  $O(\lg^{1+\epsilon} n)$  time in total.

To update  $F_{t-1}$ , let  $v$  be the  $\alpha$ th child of  $r$ . We observe that it suffices to update  $\mathcal{B}(v)$  without making changes to any other tree in  $F_{t-1}$ . Then we claim that if  $x$  is also labeled  $\alpha$  in  $\mathcal{B}(r)$ , then in  $\mathcal{B}(v)$ , we will also insert  $h$  as a child of the original node corresponding to  $x$ ; otherwise, we insert  $h$  as a child of the original node of  $\mathcal{B}(v)$  that corresponds to the node,  $x'$ , in  $\mathcal{B}(r)$  that is `parent $_{\alpha}(x)$` . If  $x'$  does not exist, then  $h$  is inserted as a child of the dummy root of  $\mathcal{B}(v)$ . We then observe that  $h$  will be inserted to  $\mathcal{B}(v)$  as the new parent of the set of children of  $x$ ,  $x'$  or the dummy root (depending on which of the above three cases applies) that are between and including the original nodes in  $\mathcal{B}(v)$  that correspond to the nodes `pre_succ $_{\alpha}(y)$`  and `post_pred $_{\alpha}(z)$`  in  $\mathcal{B}(r)$ . Thus, in  $O(\lg n / \lg \lg n)$  time, we have found where to insert  $h$  in  $F_{t-1}$ , and by the approach shown in the previous paragraph, we can use `link`, `cut` and `modify`

to update  $F_{i-1}$  in  $O(\lg^{1+\epsilon} n)$  time. This process can then be repeated at each successive level of  $W$ . Hence it requires  $O(\lg^{2+\epsilon} n / \lg \lg n)$  time to update all the  $F_i$ 's.

When updating the  $F_i$ 's, we also update inter-level pointers. The details for that are deferred to the full version.  $\square$

The space analysis is also omitted. We thus have the final result:

**Theorem 1.** *Under the word RAM model with word size  $w = \Omega(\lg n)$ , an ordinal tree on  $n$  weighted nodes can be stored in  $O(n)$  words of space, such that path reporting queries can be answered in  $O((\lg n / \lg \lg n)^2 + \text{occ} \lg n / \lg \lg n)$  time, where  $\text{occ}$  is the output size, and `modify_weight`, `node_insert` and `node_delete` can be supported in  $O(\lg^{2+\epsilon} n)$  amortized time for any constant  $\epsilon \in (0, 1)$ .*

## References

1. Alstrup, S., Holm, J.: Improved algorithms for finding level ancestors in dynamic trees. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, p. 73. Springer, Heidelberg (2000)
2. Alstrup, S., Husfeldt, T., Rauhe, T.: Marked ancestor problems. In: FOCS, pp. 534–544 (1998)
3. Arge, L., Vitter, J.S.: Optimal external memory interval management. *SIAM Journal on Computing* **32**(6), 1488–1508 (2003)
4. Bille, P.: A survey on tree edit distance and related problems. *Theor. Comput. Sci.* **337**(1–3), 217–239 (2005)
5. Brodal, G.S., Davoodi, P., Srinivasa Rao, S.: Path minima queries in dynamic weighted trees. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 290–301. Springer, Heidelberg (2011)
6. Chan, T.M., He, M., Munro, J.I., Zhou, G.: Succinct indices for path minimum, with applications to path reporting. In: Schulz, A.S., Wagner, D. (eds.) ESA 2014. LNCS, vol. 8737, pp. 247–259. Springer, Heidelberg (2014)
7. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. *Algorithmica* **2**, 337–361 (1987)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 3rd edn. MIT Press (2009)
9. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.* **14**(4), 781–798 (1985)
10. Frederickson, G.N.: Ambivalent data structures for dynamic 2-edge-connectivity and  $k$  smallest spanning trees. *SIAM J. Comput.* **26**(2), 484–538 (1997)
11. Frederickson, G.N.: A data structure for dynamically maintaining rooted trees. *J. Algorithms* **24**(1), 37–65 (1997)
12. He, M., Munro, J.I.: Space efficient data structures for dynamic orthogonal range counting. *Comput. Geom.* **47**(2), 268–281 (2014)
13. He, M., Munro, J.I., Satti, S.R.: Succinct ordinal trees based on tree covering. *ACM Transactions on Algorithms* **8**(4), 42 (2012)
14. He, M., Munro, J.I., Zhou, G.: Path queries in weighted trees. In: Asano, T., Nakano, S., Okamoto, Y., Watanabe, O. (eds.) ISAAC 2011. LNCS, vol. 7074, pp. 140–149. Springer, Heidelberg (2011)

15. He, M., Munro, J.I., Zhou, G.: Succinct data structures for path queries. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 575–586. Springer, Heidelberg (2012)
16. He, M., Munro, J.I., Zhou, G.: A framework for succinct labeled ordinal trees over large alphabets. *Algorithmica* (to appear, 2014)
17. Kaplan, H., Shafir, N.: Path minima in incremental unrooted trees. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 565–576. Springer, Heidelberg (2008)
18. Krizanc, D., Morin, P., Smid, M.H.M.: Range mode and range median queries on lists and trees. *Nord. J. Comput.* **12**(1), 1–17 (2005)
19. Navarro, G., Nekrich, Y.: Optimal dynamic sequence representations. In: SODA, pp. 865–876 (2013)
20. Navarro, G., Sadakane, K.: Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms* **10**(3), 16 (2014)
21. Patil, M., Shah, R., Thankachan, S.V.: Succinct representations of weighted trees supporting path queries. *J. Discrete Algorithms* **17**, 103–108 (2012)

# **Matching and Assignment II**



# Linear-Time Algorithms for Proportional Apportionment

Zhanpeng Cheng<sup>(✉)</sup> and David Eppstein

Department of Computer Science, University of California, Irvine, USA  
zhanpenc@uci.edu

**Abstract.** The apportionment problem deals with the fair distribution of a discrete set of  $k$  indivisible resources (such as legislative seats) to  $n$  entities (such as parties or geographic subdivisions). Highest averages methods are a frequently used class of methods for solving this problem. We present an  $O(n)$ -time algorithm for performing apportionment under a large class of highest averages methods. Our algorithm works for all highest averages methods used in practice.

## 1 Introduction

After an election, in parliamentary systems based on party-list proportional representation, the problem arises of allocating seats to parties so that each party's number of seats is (approximately) proportional to its number of votes [1]. Several methods, which we survey in more detail below, have been devised for calculating how many seats to allocate to each party. Often, these methods involve *sequential allocation* of seats under a system of priorities calculated from votes and already-allocated seats. For instance, the Sainte-Laguë method, used for elections in many countries, allocates seats to parties one at a time, at each step choosing the party that has the maximum ratio of votes to the denominator  $2s + 1$ , where  $s$  is the number of seats already allocated to the same party.

*Legislative apportionment*, although mathematically resembling seat allocation, occurs at a different stage of the political system, both in parliamentary systems and in the U.S. Congress [2, 3]. It concerns using population counts to determine how many legislative seats to allocate to each state, province, or other administrative or geographic subdivision, prior to holding an election to fill those seats. Again, many apportionment methods have been developed, some closely related to seat allocation methods. For instance, a method that generates the same results as Sainte-Laguë (calculated by a different formula) was proposed by Daniel Webster for congressional seat apportionment. However, although similar in broad principle, seat allocation and apportionment tend to differ in detail because of the requirement in the apportionment problem that every administrative subdivision have at least one representative. In contrast, in seat allocation, sufficiently small parties might fail to win any seats and indeed some seat allocation methods use artificially high thresholds to reduce the number of represented parties.

We may formalize these problems mathematically as a form of *diophantine approximation*: we are given a set of  $k$  indivisible resources (legislative seats) to be distributed to  $n$  entities (parties or administrative subdivisions), each with score  $v_i$  (its vote total or population), so that the number of resources received by an entity is approximately proportional to its score. The key constraint here is that the entities can only receive an integral amount of resources; otherwise, giving the  $i$ th entity  $kv_i/\sum v_i$  units of resource solves the problem optimally. Outside of political science, forms of the apportionment problem also appear in statistics in the problem of rounding percentages in a table so that they sum to 100% [4] and in manpower planning to allocate personnel [5].

Broadly, most apportionment methods can be broken down into two classes: *largest remainder methods* in which a fractional solution to the apportionment problem is rounded down to an integer solution, and then the remaining seats are apportioned according to the distance of the fractional solution from the integer solution, and *highest averages methods* like the Sainte-Laguë method described above, in which seats are assigned sequentially prioritized by a combination of their scores and already-assigned seats. Largest remainder methods are trivial from the algorithmic point of view, but are susceptible to certain electoral paradoxes. Highest averages methods avoid this problem, and are more easily modified to fit different electoral circumstances, but appear a priori to be slower. When implemented naively, they might take as much as  $O(n)$  time per seat, or  $O(nk)$  overall. Priority queues can generally be used to reduce this naive bound to  $O(\log n)$  time per seat, or  $O(k \log n)$  overall [6], but this is still suboptimal, especially when there are many more seats than parties ( $k \gg n$ ). We show here that many of these methods can be implemented in time  $O(n)$ , an optimal time bound as it matches the input size. We do not expect this speedup to have much effect in actual elections, as the time to compute results is typically minuscule relative to the time and effort of conducting an election; however, the speedup we provide may be of benefit in simulations, where a large number of simulated apportionment problems may need to be solved in order to test different variations in the parameters of the election system or a sufficiently large sample of projected election outcomes.

### 1.1 Highest Averages

We briefly survey here highest averages methods (or Huntington methods), a class of methods used to solve the apportionment problem [7,8]. Balinski and Young [9] showed that divisor methods, a subclass of the highest averages methods, are the only apportionment methods that avoid undesirable outcomes such as the *Alabama paradox*, in which increasing the number of seats to be allocated can cause a party's individual allocation to decrease. Because they avoid problematic outcomes such as this one, almost all apportionment methods in use are highest averages methods.

In a highest averages method, a sequence of divisors  $d_0, d_1, \dots$  is given as part of the description of the method and determines the method. To apportion the resources, each entity is assigned an initial priority  $v_i/d_0$ . The entity with the

highest priority is then given one unit of resource and has its priority updated to use the next divisor in the sequence (i.e., if the winning entity  $i$  is currently on divisor  $d_j$ , then its priority is updated as  $v_i/d_{j+1}$ ). This process repeats until all the resources have been exhausted. The priorities  $v_i/d_j$  are also called *averages*, giving the method its name. Table 1 gives the sequence of divisors for several common highest averages methods.

**Table 1.** Divisors for common highest averages methods

Method	Other Names	Divisors
Adams	Smallest divisors	$0, 1, 2, 3, \dots, j, \dots$
Jefferson	Greatest divisors, d'Hondt	$1, 2, 3, 4, \dots, j + 1, \dots$
Sainte-Laguë	Webster, Major fractions	$1, 3, 5, 7, \dots, 2j + 1, \dots$
Modified Sainte-Laguë	—	$1.4, 3, 5, 7, \dots$
Huntington–Hill	Equal proportions, Geometric mean	$0, \sqrt{2}, \sqrt{6}, \dots, \sqrt{j(j+1)}, \dots$
Dean	Harmonic mean	$0, 4/3, 12/5, \dots, \frac{2a(a+1)}{2a+1}, \dots$
Imperiali	—	$2, 3, 4, 5, \dots, j + 2, \dots$
Danish	—	$1, 4, 7, 10, \dots, 3j + 1, \dots$

A zero at the start of the divisor sequence prioritizes the first assignment to each entity over any subsequent assignment, in order to ensure that (if possible) every entity is assigned at least one unit. If a zero is given, but the number of units is less than the number of entities, the entities are prioritized by their  $v_i$  values.

### 1.2 New Results

In this paper, we present an  $O(n)$ -time algorithm for simulating a highest averages method. Our algorithm works only for divisor sequences that are close to arithmetic progressions; however, this includes all methods used in practice, since this property is necessary to achieve approximately-proportional apportionment. For divisor sequences that are already arithmetic progressions, our algorithm transforms the problem into finding the  $k$ th smallest value in the disjoint union of  $n$  implicitly defined arithmetic progressions, which we solve in  $O(n)$  time. For methods with divisor sequences close to but not equal to arithmetic progressions, we use an arithmetic progression to approximate the divisor sequence, and show that this still gives us the desired result.

### 1.3 Related Work

An alternative view of a number of highest averages methods is to find a multiplier  $\lambda > 0$  such that  $\sum_i [\lambda v_i] = k$ , where  $[\cdot]$  is a suitable rounding function for the method. The  $i$ th entity is then apportioned the amount  $[\lambda v_i]$ . For example,

the standard rounding function gives rise to the Sainte-Laguë method and the floor function gives rise to the Adam method. For these methods, the problem can be solved in  $O(n^2)$  time, or  $O(n \log n)$  with a priority queue [10, 11]<sup>1</sup>. The algorithm works by initializing  $\lambda = k / \sum_j v_j$  and iteratively choosing a new apportionment that reduces the difference between  $\sum_i \lfloor \lambda v_i \rfloor$  and  $k$ . The number of new apportionments can be shown to be at most  $n$ .

Selecting the  $k$ th smallest element in certain other types of implicitly defined sets has also been well studied. Gagil and Megiddo studied the assignment of  $k$  workers to  $n$  jobs, where the implicitly defined sets are induced by concave functions giving the utility of assigning  $k_i$  workers to job  $i$  [14]. Their  $O(n \log^2 k)$  algorithm was improved by Frederickson and Johnson to  $O(n + p \log(k/p))$  where  $p = \min(k, n)$  [15]. For implicit sets given as an  $n \times m$  matrix with sorted rows and columns, Frederickson and Johnson found an  $O(h \log(2k/h^2))$  time algorithm, where  $h = \min(\sqrt{k}, m)$  and  $m \leq n$  [16]. Sorting the inputs would turn our problem into sorted matrix selection, but the  $O(n \log n)$  sorting time would already exceed our time bound.

## 2 Preliminaries

Given strictly increasing divisors  $d_0, d_1, d_2, \dots$ , our goal is to simulate the highest averages method induced by those divisors in time linear in the number of entities. Instead of directly selecting the entities with the  $k$  largest priorities, we take advantage of the arithmetic progression structure of the divisors and consider the problem as selecting the  $k$  smallest inverted priorities. Associate the  $i$ th entity to the increasing sequence

$$A_i = \left\{ \frac{d_j}{v_i} : j = 0, 1, 2, \dots \right\}.$$

Let  $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$  and  $U(\mathcal{A})$  be the multiset formed from the disjoint union of the sequences. The problem is to find the value of the  $k$ th smallest element of  $U(\mathcal{A})$ .

We do not actually produce the  $k$  smallest elements, only the value of the  $k$ th smallest one, allowing us to eliminate any dependence on  $k$  in our time bounds. An explicit list of the  $k$  smallest elements is also not necessary for the election problem, since we are primarily interested in the total amount of resources allocated to each entity, which can be calculated from the value of the  $k$ th smallest element. When a rank function (defined in the following paragraph) can be computed in constant time, we may use it to compute, in constant time for each  $A_i$ , the largest index  $j$  such that  $d_j/v_i$  is at most the computed value, which gives

---

<sup>1</sup> An anonymous reviewer suggested that linear-time algorithms were given previously in two Japanese papers [12, 13]. However, we were unable to track down these papers nor could we determine whether their time was linear in the number of votes, seats, or parties. The second reference, in particular, does not appear to be on the IEICE website, neither searching by year and page number nor with broad search terms.

the allocation to entity  $i$ . Producing only the value of the  $k$ th smallest element also sidesteps the issue of tie-breaking when several entities have the same priorities and are equally eligible for the last resource. The rules for breaking ties are application-dependent, so it is best to leave them out of the main algorithm.

For a sequence  $A$ , let  $A(j)$  denote the  $j$ th element of the sequence  $A$ , with the first element at index 0. We let  $A(-1) = -\infty$  to avoid corner cases in the algorithm; however, when counting elements of  $A$ , this  $-\infty$  value should be ignored. Define the *rank* of a number  $x$  in  $A$  as the number of elements of  $A$  less than or equal to  $x$ . Equivalently, for a strictly monotonic sequence, this is the index  $j$  such that  $A(j) \leq x < A(j + 1)$ . We denote the rank function by  $r(x, A)$ . For a set  $\mathcal{A}$  of sequences, we define  $r(x, \mathcal{A}) = \sum_{A \in \mathcal{A}} r(x, A)$ . If the set  $\mathcal{A}$  is clear from context, we will drop  $\mathcal{A}$  and simply write  $r(x)$ .

In our algorithm, we assume that the rank function for each sequence  $A_i$  can be computed in constant time. When the sequences are arithmetic progressions (as they are in most of the voting methods we consider), these functions can be computed using only a constant number of basic arithmetic operations, so this is not a restrictive assumption. The Huntington–Hill method involves square roots, but its rank function may still be calculated using a constant number of operations that are standard enough to be included as hardware instructions on modern processors.

Observe a small subtlety about the rank function: if  $\tau$  is the  $k$ th smallest element, then  $r(\tau)$  is not necessarily  $k$ . Indeed,  $r(\tau)$  can be greater than  $k$ , as in the case where there are  $k - 1$  elements of  $U(\mathcal{A})$  less than  $\tau$  and  $\tau$  is duplicated twice, in which case  $r(\tau) = k + 1$ . In general, we have  $k \leq r(\tau) \leq k + n - 1$ . The rank of the  $k$ th smallest element can still be characterized, through the following observation.

**Observation 1.**  $\tau$  is the value of the  $k$ th smallest element in  $U(\mathcal{A})$  if and only if  $r(\tau) \geq k$  and for all  $x < \tau$ ,  $r(x) < k$ .

Now define  $L(x, A)$  as the largest value in  $A$  less than  $x$ ; similarly, define  $G(x, A)$  as the smallest value in  $A$  greater than  $x$ . Note that  $L(x, A)$  and  $G(x, A)$  can be computed easily from the rank of  $x$ : if  $r = r(x, A)$ , then  $L(x, A)$  and  $G(x, A)$  must be the value of either  $A(r - 1)$ ,  $A(r)$ , or  $A(r + 1)$ . For  $\mathcal{A}$ , we use the similar notation  $L(x, \mathcal{A})$  (respectively  $G(x, \mathcal{A})$ ) to denote the multiset of  $L(x, A)$  (respectively  $G(x, A)$ ) over all  $A \in \mathcal{A}$ .

Lastly, we make note of one notational convention. In our descriptions, the input variables to an algorithm may change within the algorithm, and it is useful to talk about both the values of the variables as they change and their initial values. Therefore, we use a tilde to denote the initial value of a variable, and the lack of a tilde to denote the changing value over the course of the algorithm. For example,  $\tilde{\mathcal{A}}$  means the initial value and  $\mathcal{A}$  means the value at an intermediate point of the algorithm.

---

**Algorithm 1.** LOWERRANKCOARSE SOLUTION( $\tilde{\mathcal{A}}, \tilde{k}, \xi$ )

---

**Input:**  $\tilde{\mathcal{A}}$ : set of increasing sequences;  $\tilde{k}$ : positive integer;  $\xi$ : a coarse solution with  $r(\xi, \tilde{\mathcal{A}}) \geq k$

**Output:** another coarse solution  $\xi'$  with  $r(\xi', \tilde{\mathcal{A}}) < k$

```

1:  $\mathcal{A} \leftarrow \tilde{\mathcal{A}}, k \leftarrow \tilde{k}, u \leftarrow \xi$ 
2: loop
3:    $\bar{x} \leftarrow$  median of  $L(u, \mathcal{A})$ 
4:   if  $r(\bar{x}, \mathcal{A}) \geq k$  then
5:      $u \leftarrow \bar{x}$ 
6:   else if  $r(\bar{x}, \mathcal{A}) < k - n$  then
7:      $\mathcal{B} \leftarrow \{A : A \in \mathcal{A} \text{ and } L(u, A) \leq \bar{x}\}$ 
8:      $k \leftarrow k - \sum_{A \in \mathcal{B}} r(\bar{x}, A)$ 
9:      $\mathcal{A} \leftarrow \mathcal{A} \setminus \mathcal{B}$ 
10:  else
11:    return  $\bar{x}$ 
12:  end if
13: end loop

```

---

### 3 The Algorithm

Our algorithm has three parts. In the first part, we show how the value of the  $k$ th smallest element of  $U(\mathcal{A})$  can be found from a *coarse solution*, a value whose rank is within  $O(n)$  positions of  $k$ . In the second part, we handle a special case of the problem in which every sequence in  $\mathcal{A}$  is an arithmetic sequence, by showing how the rank function over  $\mathcal{A}$  can be inverted in this case to produce a coarse solution. And in the last part, we deal with more general sequences, by showing how arithmetic sequences that approximate them can be used to produce a coarse solution.

#### 3.1 From Coarse to Exact Solutions

In this section, we show how to compute the value of the  $k$ th smallest element of  $U(\mathcal{A})$ , given a coarse solution. A value  $\xi$  is called a *coarse solution* for  $k$  if  $|k - r(\xi)| \leq cn$  for some constant  $c$ . Equivalently, this means there are only  $O(n)$  elements between  $\xi$  and the  $k$ th smallest element. Note that  $\xi$  does not have to be an element of  $U(\mathcal{A})$ .

Before presenting the algorithm, we first show that the coarse solution can be assumed to have a rank smaller than  $k$ .

**Lemma 1.** *Let  $\xi$  be a coarse solution for  $k$ , and assume  $r(x, A)$  can be computed in constant time for every  $A \in \mathcal{A}$ . If  $r(\xi, \mathcal{A}) \geq k$ , then another coarse solution  $\xi'$  with  $r(\xi', \mathcal{A}) < k$  can be found in  $O(n)$  time.*

*Proof.* We find a  $\xi'$  such that  $\tilde{k} - n \leq r(\xi', \tilde{\mathcal{A}}) < \tilde{k}$ . To find this value, start with  $u = \xi$ . Then, repeatedly update  $u$  and  $\mathcal{A}$  as follows, until the median  $\bar{x}$  of  $L(u, \mathcal{A})$  has rank between  $k - n$  and  $k$ :

1. If  $r(\bar{x}, \mathcal{A}) \geq k$ , then set  $u = \bar{x}$ .
2. If  $r(\bar{x}, \mathcal{A}) < k - n$ , then any sequence  $A$  in  $\mathcal{A}$  with  $L(u, A) \leq \bar{x}$  can no longer help us get closer to a value in the desired range, so we remove those sequences and update  $k$  accordingly to compensate for their removal (i.e., subtract from  $k$  the ranks  $r(\bar{x}, A)$  over all removed sequences  $A$ ).

Algorithm 1 summarizes this procedure.

Let  $q$  be the sum of two quantities: the distance from the rank of  $u$  to  $\tilde{k}$ , and the number of sequences remaining in  $\mathcal{A}$ . Then  $q$  is initially  $O(n)$  by the assumption that  $\tilde{\xi}$  is a coarse solution. Each iteration of the loop of the algorithm takes time  $O(|\mathcal{A}|)$  and reduces  $q$  by  $O(|\mathcal{A}|)$  units, either by reducing the rank of  $u$  in the first case or by eliminating sequences from  $\mathcal{A}$  in the second case. Eventually (before  $q$  can be reduced to zero) the algorithm must terminate, at which point it has taken time proportional to the total reduction in  $q$ , which is  $O(n)$ . When it terminates, the returned value is clearly a coarse solution whose rank is less than  $k$ , as desired. □

We now present the algorithm to convert a coarse solution to an exact one. The algorithm is similar to a binary search, where we maintain both a lower bound and an upper bound that narrow the possible candidates as the algorithm progresses. The lower bound is initially derived from the coarse solution, which guarantees that it is close to the true solution. The main difference between our algorithm and a standard binary search is that we do not know the distribution of the sequences' elements within the bounds, so we cannot reduce search space by a constant proportion simply by splitting the range halfway between the lower and upper bounds. Instead, we split on the median of some well-chosen set within the bounds, which will allow us to reduce the number of candidates by at least  $|\mathcal{A}|/2$  in each step. To make the algorithm run in linear time, sequences that no longer have elements between the bounds are removed from  $\mathcal{A}$ . The procedure is similar to the one in Algorithm 1. But instead of moving down in  $U(\mathcal{A})$  with  $L(\cdot, \mathcal{A})$ , the algorithm moves up with  $G(\cdot, \mathcal{A})$ . Because of this, compensating  $k$  when removing a sequence is no longer as straight-forward. In particular, we may need to query the rank of the upper bound  $u$ , so we need an extra variable to keep track of the possible under-compensation to the rank for these values. The details are presented in the theorem below.

**Theorem 2.** *Let  $\mathcal{A}$  be a set of increasing sequences. Assume  $r(x, A)$  can be computed in constant time for every  $A \in \mathcal{A}$ . If a coarse solution  $\xi$  is given, then the value of the  $k$ th smallest element in  $U(\mathcal{A})$  can be found in  $O(n)$  time.*

For space reasons we defer a detailed proof to the full paper.

### 3.2 Coarse Solution for Arithmetic Sequences

In this section, we focus on the special case where every sequence in  $\mathcal{A}$  is an arithmetic sequence. In particular, each sequence  $A$  is of the form  $A(j) = x_A + y_A \cdot j$

---

**Algorithm 2.** COARSETOEXACT( $\tilde{\mathcal{A}}, \tilde{k}, \xi$ )

---

**Input:**  $\tilde{\mathcal{A}}$ : set of increasing sequences;  $\tilde{k}$ : positive integer;  $\xi$ : coarse solution with  $r(\xi, \tilde{\mathcal{A}}) < k$

**Output:** the value of the  $\tilde{k}$ th smallest element in  $U(\tilde{\mathcal{A}})$

```

1:  $\mathcal{A} \leftarrow \tilde{\mathcal{A}}, k \leftarrow \tilde{k}$ 
2:  $l \leftarrow \xi, u \leftarrow \infty, m \leftarrow 0$ 
3: repeat
4:    $\bar{x} \leftarrow$  median of  $G(l, \mathcal{A})$ 
5:   if  $r(\bar{x}, \mathcal{A}) < k$  then
6:      $l \leftarrow \bar{x}$ 
7:   else
8:      $u \leftarrow \bar{x}$ 
9:      $m \leftarrow 0$ 
10:  end if
11:  if  $|\{A : G(l, A) = u\}| \geq 1$  then
12:     $m \leftarrow m + |\{A : G(l, A) = u\}| - 1$ 
13:  end if
14:   $\mathcal{A}' \leftarrow \{A : A \in \mathcal{A} \text{ and } G(l, A) < u\} \cup \{\text{any one } A \in \mathcal{A} \text{ such that } G(l, A) = u\}$ 
15:   $k \leftarrow k - \sum_{A \in \mathcal{A} \setminus \mathcal{A}'} r(l, A)$ 
16:   $\mathcal{A} \leftarrow \mathcal{A}'$ 
17: until  $G(l, \mathcal{A})$  has only one value  $t$  and ( $r(t, \mathcal{A}) \geq k$  or ( $t = u$  and  $r(t, \mathcal{A}) \geq k - m$ 
)
18: return  $t$ 

```

---

with  $y_A > 0$ , for  $j = 0, 1, 2, \dots$ . In this case, the rank function is given by

$$r(x) = \sum_{A \in \mathcal{A}} r(x, A) = \sum_{A \in \mathcal{A}} \left( 1 + \left\lfloor \frac{x - x_A}{y_A} \right\rfloor \right) I_{x \geq x_A} \tag{1}$$

where  $I_{x \geq x_A}$  is the indicator function that is 1 when  $x \geq x_A$  and 0 otherwise.

We can think of the problem of finding the value of the  $k$ th smallest element as “inverting” the rank function to find  $x$  such that  $r(x)$  is close to  $k$ . Of course, the inverse of  $r(\cdot)$  does not make sense, since the function is neither one-to-one nor onto  $\mathbb{N}$ . However, if we drop the floor and the plus one, and consider

$$s(x) = s(x, \mathcal{A}) = \sum_{A \in \mathcal{A}} s(x, A) = \sum_{A \in \mathcal{A}} \left( \frac{x - x_A}{y_A} \right) I_{x \geq x_A}, \tag{2}$$

the resulting function is piecewise linear with a well-defined inverse.

Call the sequence  $A$  *contributing* for  $k$  if  $s(A(0)) = s(x_A) \leq k$ . To invert  $s(x)$ , we need to first find the contributing sequences of  $\mathcal{A}$  for  $k$ . This can be done in  $O(n)$  time, as shown in the next lemma.

**Lemma 2.** *Suppose  $\mathcal{A}$  is a set of arithmetic sequences of the form  $A(j) = x_A + y_A \cdot j$  with  $y_A > 0$ , then the contributing sequences of  $\mathcal{A}$  can be found in  $O(n)$  time.*



**Algorithm 3.** FINDCONTRIBUTINGSEQUENCES( $\tilde{\mathcal{A}}, k$ )

**Input:**  $\tilde{\mathcal{A}}$ : set of arithmetic sequences of the form  $A(j) = x_A + y_A j$ ;  $k$ : integer  $> n$   
**Output:** subset  $C$  of  $\tilde{\mathcal{A}}$  of contributing sequences for  $k$

- 1:  $\mathcal{A} \leftarrow \tilde{\mathcal{A}}, C \leftarrow \emptyset.$
- 2: **while**  $\mathcal{A}$  is nonempty **do**
- 3:    $\bar{x} \leftarrow$  median of  $\{x_A : A \in \mathcal{A}\}$
- 4:   **if**  $s(\bar{x}, \tilde{\mathcal{A}}) > k$  **then**
- 5:      $\mathcal{A} \leftarrow \{A : A \in \mathcal{A} \text{ and } x_A < \bar{x}\}$
- 6:   **else**
- 7:      $C \leftarrow C \cup \{A : A \in \mathcal{A} \text{ and } x_A \leq \bar{x}\}$
- 8:      $\mathcal{A} \leftarrow \{A : A \in \mathcal{A} \text{ and } x_A > \bar{x}\}$
- 9:   **end if**
- 10: **end while**
- 11: **return**  $C$

*Proof.* The algorithm to find the contributing sequences is given in Algorithm 3. In the main loop, we repeatedly compute the median  $\bar{x}$  of the first element of remaining sequences  $\mathcal{A}$ . Comparison of  $s(\bar{x}, \tilde{\mathcal{A}})$  to  $k$  then eliminates a portion of those sequences and determines which of the eliminated sequences are contributing:

1. If  $s(\bar{x}, \tilde{\mathcal{A}}) > k$ : None of the sequences with  $x_A \geq \bar{x}$  can be contributing, so we reduce  $\mathcal{A}$  to only those with  $x_A < \bar{x}$ .
2. If  $s(\bar{x}, \tilde{\mathcal{A}}) \leq k$ : The sequences with  $x_A \leq \bar{x}$  are contributing, so we add them all to the list  $C$ , and reduce  $\mathcal{A}$  to only those sequences with  $x_A > \bar{x}$ .

The loop repeats until  $\mathcal{A}$  is empty, at which time  $C$  is the output.

The algorithm’s correctness follows from the fact that a sequence is added to  $C$  if and only if it is contributing. For the algorithm to run in  $O(n)$  time, each iteration of the loop must run in  $O(|\mathcal{A}|)$  time. In particular, we must compute  $s(\bar{x}, \tilde{\mathcal{A}}) = s(\bar{x}, \mathcal{A}) + s(\bar{x}, C)$  in  $O(|\mathcal{A}|)$  time. To do this, note that when a sequence  $A$  is added to  $C$ , all subsequent  $\bar{x}$  have  $\bar{x} > x_A$ . This means we can consider  $s(\cdot, C)$  as a linear function, whose value can be computed in constant time by keeping track of the two coefficients of the linear function and updating them whenever sequences are added to  $C$ .  $\square$

Once the contributing sequences  $C$  are found, we can restrict our search for  $x$  to the last interval of  $s(\cdot, C)$ , which is a linear function without breakpoints and can be inverted easily. In particular, the inverse is given by

$$s^{-1}(k) = \left( \sum_{A \in C} \frac{1}{y_A} \right)^{-1} \left( k + \sum_{A \in C} \frac{x_A}{y_A} \right) \tag{3}$$

which can be interpreted as a weighted and shifted harmonic mean of the slopes of contributing arithmetic sequences. Since  $r(x) - s(x) \leq n$ ,  $s^{-1}(k)$  is a coarse solution, applying [Theorem 2](#) immediately gives the following.

**Theorem 3.** *If  $\mathcal{A}$  is a set of arithmetic sequences, then the value of the  $k$ th smallest element in  $U(\mathcal{A})$  can be found in  $O(n)$  time.*

While not necessary for subsequent sections, a slight generalization can be made here. The main property of arithmetic sequences we used is that  $r(x, A)$  is approximable by a function that can be written as  $w_A f(x) + u_A$ , where  $f$  is an invertible function independent of  $A$ ;  $w_A, u_A$  are constants; and  $w_A > 0$ . When  $r(x, A)$  can be approximated in this way, the inverse is given by:

$$s^{-1}(k) = f^{-1} \left( \frac{k - \sum_{A \in \mathcal{C}} u_A}{\sum_{A \in \mathcal{C}} w_A} \right) \tag{4}$$

where  $\mathcal{C}$  is the set of contributing sequences. For example, if  $A(j) = 2^j/v_A$ , then the rank function is given by  $r(x, A) = \lfloor \log(xv_A) \rfloor$  and is approximable by  $\log x + \log v_A$ . The coarse solution is then given by

$$s^{-1}(k) = e^{\frac{1}{|\mathcal{C}|}} (k - \sum_{A \in \mathcal{C}} \log v_A). \tag{5}$$

### 3.3 Coarse Solution for Approximately-Arithmetic Sequences

In this section, we show how to handle more general sequences for highest averages methods. Our strategy works as long as the divisor sequence  $D = \{d_j\}$  is close to an arithmetic progression  $E = \{e_j\}$ , where closeness here means that there is a constant  $c$  so that  $|d_j - e_j| \leq c$  for every  $j \geq 0$ .

Suppose  $\mathcal{A}$  is the set of (arithmetic) sequences induced by the arithmetic divisor sequence  $E$  and  $\mathcal{B}$  is the set of sequences induced by the divisor sequence  $D$ . We show that, if  $E$  and  $D$  are close, then the rank of every number  $x$  in  $\mathcal{A}$  is within a constant of the rank in  $\mathcal{B}$ . This means that a coarse solution for  $\mathcal{A}$  is also a coarse solution for  $\mathcal{B}$ . Hence, to do apportionment for these more general sequences, we just use the results from the previous section to find a coarse solution for the approximating arithmetic sequences, then apply [Theorem 2](#) to the original sequences with that coarse solution.

**Lemma 3.** *Let  $A$  be an arithmetic progression  $A(j) = x_A + y_A j$  with slope  $y_A > 0$ , and let  $B$  be an increasing sequence such that for every  $j$ ,  $|A(j) - B(j)| \leq cy_A$  for some constant  $c$ . Then  $|r(x, A) - r(x, B)| \leq c'$  for another constant  $c'$ .*

*Proof.* Fix  $x$ . Let  $a = r(x, A)$  and  $b = r(x, B)$ . Note that  $A(a)$  (respectively  $B(b)$ ) is largest value of  $A$  (respectively  $B$ ) less than or equal to  $x$ . We have two cases:

**Case  $A(b) \geq x$ :** Note that  $a \leq b$  since  $A(a) \leq x \leq A(b)$ . Furthermore,

$$A(b) - A(a) \leq [A(b) - B(b)] + [x - A(a)] \leq cy_A + y_A.$$

This means that there are at most  $c + 1$  elements of  $A$  between  $A(a)$  and  $A(b)$ , so  $b - a \leq c + 1$ .

**Case  $A(b) < x$ :** Note that  $b \leq a$ . Furthermore,

$$\begin{aligned} x - B(b) &< B(b + 1) - B(b) \\ &\leq |B(b + 1) - A(b + 1)| + |A(b + 1) - A(b)| + |A(b) - B(b)| \\ &\leq cy_A + y_A + cy_A = (2c + 1)y_A \end{aligned}$$

It follows that  $A(a) - A(b) \leq [x - B(b)] + |B(b) - A(b)| \leq (2c + 1)y_A + cy_A$ , and  $a - b \leq 3c + 1$ .

In both cases, we have  $|r(x, A) - r(x, B)| = |b - a| \leq \text{const}$ , as needed. □

From the lemma, the following is immediate.

**Theorem 4.** *Let the divisor sequence  $E = \{e_j\}$  be an arithmetic progression, and suppose  $D = \{d_j\}$  is another divisor sequence such that for every  $j \geq 0$ ,  $|d_j - e_j| \leq c$  for some constant  $c$ . If  $\mathcal{A}$  is the set of sequences induced by  $E$  and  $\mathcal{B}$  the set of sequences induced by  $D$ , then for every  $x$ ,  $|r(x, \mathcal{A}) - r(x, \mathcal{B})| \leq c'n$ , for some constant  $c'$ .*

*Proof.* Let  $A \in \mathcal{A}$  and  $B$  be the corresponding sequence in  $\mathcal{B}$  with score  $v$ . We have

$$|A(j) - B(j)| = \frac{|d_j - e_j|}{v} \leq \frac{c}{v}.$$

Note that the slope of  $A$  is  $1/v$ , so by Lemma 3,  $|r(x, A) - r(x, B)| \leq c'$  for some constant  $c'$ . Summing over all sequences gives the desired result. □

The strategy in this section works for all the methods in Table 1 with non-arithmetic divisor sequences. The Huntington–Hill and Dean methods have divisor sequences where each element of the sequence is the geometric or harmonic mean respectively of consecutive natural numbers. Hence, each element is within 1 of the arithmetic sequence  $d_j = j$ , so finding a coarse solution under the divisor sequence  $d_j = j$  is enough to find a coarse solution for those two methods. For methods like the modified Sainte-Laguë method, where the divisor sequence is arithmetic except for a constant number of elements, we may ignore the non-arithmetic elements and find a coarse solution solely from the remaining elements that do form an arithmetic progression.

## 4 Conclusion

We have shown that many commonly used apportionment methods can be implemented in time linear in the size of the input (vote totals for each entity), based on a transformation of the problem into selection in multisets formed from unions of arithmetic or near-arithmetic sequences. Our method can be extended to selection in unions of other types of sequences, as long as the rank functions of the sequences can be approximately inverted and aggregated. It would be of interest to determine whether forms of diophantine approximation from other application areas can be computed as efficiently.

**Acknowledgments.** This research was supported in part by NSF grant 1228639, and ONR grant N00014-08-1-1015.

## References

1. Lijphart, A.: Degrees of Proportionality of Proportional Representation Formulas. In: Grofman, B., Lijphart, A. (eds.) *Electoral Laws and Their Political Consequences*, pp. 170–179. Agathon Press Inc., New York (1986)
2. Athanasopoulos, B.: The Apportionment Problem and its Application in Determining Political Representation. *Spoudai Journal of Economics and Business* **43**(3–4), 212–237 (1993)
3. Owens, F.W.: On the Apportionment of Representatives. *Quarterly Publications of the American Statistical Association* **17**(136), 958–968 (1921)
4. Athanasopoulos, B.: Probabilistic Approach to the Rounding Problem with Applications to Fair Representation. In: Anastassiou, G., Rachev, S.T. (eds.) *Approximation, Probability, and Related Fields*, pp. 75–99. Springer US (1994)
5. Mayberry, J.P.: Allocation for Authorization Management (1978)
6. Campbell, R.B.: The Apportionment Problem. In: Michaels, J.G., Rosen, K.H. (eds.) *Applications of Discrete Mathematics, Updated Edition*, pp. 2–18. McGraw-Hill Higher Education, New York (2007)
7. Huntington, E.V.: The Apportionment of Representatives in Congress. *Transactions of the American Mathematical Society* **30**(1), 85–110 (1928)
8. Balinski, M.L., Young, H.P.: On Huntington Methods of Apportionment. *SIAM Journal on Applied Mathematics* **33**(4), 607–618 (1977)
9. Balinski, M.L., Young, H.P.: *Fair Representation: Meeting the Ideal of One Man, One Vote*. Yale University Press, New Haven and London (1982)
10. Dorfleitner, G., Klein, T.: Rounding with multiplier methods: An efficient algorithm and applications in statistics. *Statistical Papers* **40**(2), 143–157 (1999)
11. Zachariasen, M.: Algorithmic aspects of divisor-based biproportional rounding. Technical report, Dept. of Computer Science, University of Copenhagen (June 2005)
12. Ito, A., Inoue, K.: Linear-time algorithms for apportionment methods. In: *Proceedings of EATCS/LA Workshop on Theoretical Computer Science*, University of Kyoto, Japan, pp. 85–91 (February 2004)
13. Ito, A., Inoue, K.: On d’hondt method of computing. *IEICE Transactions D*, 399–400 (February 2006)
14. Galil, Z., Megiddo, N.: A Fast Selection Algorithm and the Problem of Optimum Distribution of Effort. *Journal of the ACM* **26**(1), 58–64 (1979)
15. Frederickson, G.N., Johnson, D.B.: The Complexity of Selection and Ranking in  $X + Y$  and Matrices with Sorted Columns. *Journal of Computer and System Sciences* **24**(2), 197–208 (1982)
16. Frederickson, G.N., Johnson, D.B.: Generalized Selection and Ranking: Sorted Matrices. *SIAM Journal on Computing* **13**(1), 14–30 (1984)

# Rank-Maximal Matchings – Structure and Algorithms

Pratik Ghosal<sup>1</sup>, Meghana Nasre<sup>2</sup>(✉), and Prajakta Nimbhorkar<sup>3</sup>

<sup>1</sup> University of Wrocław, Wrocław, Poland  
pratikghosal20082@gmail.com

<sup>2</sup> Indian Institute of Technology, Madras, India  
meghana@cse.iitm.ac.in

<sup>3</sup> Chennai Mathematical Institute, Chennai, India  
prajakta@cmi.ac.in

**Abstract.** Let  $G = (\mathcal{A} \cup \mathcal{P}, E)$  be a bipartite graph where  $\mathcal{A}$  denotes a set of agents,  $\mathcal{P}$  denotes a set of posts and ranks on the edges denote preferences of the agents over posts. A matching  $M$  in  $G$  is rank-maximal if it matches the maximum number of applicants to their top-rank post, subject to this, the maximum number of applicants to their second rank post and so on.

In this paper, we develop a *switching graph* characterization of rank-maximal matchings, which is a useful tool that encodes all rank-maximal matchings in an instance. The characterization leads to simple and efficient algorithms for several interesting problems. In particular, we give an efficient algorithm to compute the set of *rank-maximal pairs* in an instance. We show that the problem of counting the number of rank-maximal matchings is  $\#P$ -Complete and also give an FPRAS for the problem. Finally, we consider the problem of deciding whether a rank-maximal matching is *popular* among all the rank-maximal matchings in a given instance, and give an efficient algorithm for the problem.

## 1 Introduction

We consider the problem of matching applicants to posts where applicants have preferences over posts. This problem is motivated by several important real-world applications like allocation of graduates to training positions [5] and families to government housing [16]. The input to the problem is a bipartite graph  $G = (\mathcal{A} \cup \mathcal{P}, E)$ , where  $\mathcal{A}$  is a set of applicants,  $\mathcal{P}$  is a set of posts, and the set  $E$  can be partitioned as  $E = E_1 \cup \dots \cup E_r$ , where  $E_i$  contains the edges of rank  $i$ . An edge  $(a, p) \in E_i$  if  $p$  is an  $i$ th choice of  $a$ . An applicant  $a$  prefers a post  $p$  to  $p'$  if,

---

Meghana Nasre: Part of the work done by the author was supported by IIT-M initiation grant CSE/14-15/824/NFIG/MEGA.

Prajakta Nimbhorkar: Part of the work has been done while the author was on a sabbatical to the Institute of Mathematics of the Czech Academy of Sciences, Prague.

for some  $i < j$ ,  $(a, p) \in E_i$  and  $(a, p') \in E_j$ . Applicant  $a$  is indifferent between  $p$  and  $p'$  if  $i = j$ . This ranking of posts by an applicant is called *the preference list* of the applicant. When applicants can be indifferent between posts, preference lists are said to contain ties, else preference lists are strict.

The problem of matching under one-sided preferences has received lot of attention and there exist several notions of optimality like pareto-optimality [1], rank-maximality [7], popularity [2], and fairness. We focus on the well-studied notion of *rank-maximal* matchings which are guaranteed to exist in any instance. Rank-maximality was first studied under the name of *greedy matchings* by Irving [6], who also gave an algorithm for computing such matchings in case of strict lists. A rank-maximal matching matches maximum number of applicants to their rank 1 posts, subject to that, maximum number of applicants to their rank 2 posts and so on. Given an instance of the rank-maximal matchings problem possibly involving ties, Irving et al. [7] gave an  $O(\min(n + r, r\sqrt{n})m)$  time algorithm to compute a rank-maximal matching. Here  $n = |\mathcal{A}| + |\mathcal{P}|$ ,  $m = |E|$ , and  $r$  denotes the maximal rank in the instance. The weighted and capacitated versions of this problem have been studied in [11] and [14] respectively.

In this paper, we study the structure of the rank-maximal matchings using the notion of a *switching graph*. This notion was introduced in the context of *popularity* which is an alternative criterion of optimality in the one-sided preferences model. See [2] for a definition of popular matchings. McDermid and Irving [12] studied the switching graph of popular matchings for strict instances, and Nasre [13] extended it to the case of ties. This characterization has turned out to be useful for several problems like counting the number of popular matchings in strict instances, computing an *optimal* popular matching, developing an optimal manipulation strategy for an agent etc.

It is natural to extend the switching graph characterization to analyze rank-maximal matchings. Besides being interesting in its own right, it turns out to be useful in answering several natural questions. For instance, given instance  $G = (\mathcal{A} \cup \mathcal{P}, E)$ , is there a rank-maximal matching in  $G$  which matches an applicant  $a$  to a particular post  $p$ ? Is a rank-maximal matching preferred by a majority of applicants over other rank-maximal matchings in the instance? We show the following new results in this paper:

- A switching graph characterization of the rank-maximal matchings problem, and its properties, using which, we answer the questions mentioned above.
- An efficient algorithm for computing the set of *rank-maximal pairs*. An edge  $(a, p) \in E$  is a rank-maximal pair if there exists a rank-maximal matching in  $G$  that matches  $a$  to  $p$ .
- We show that the problem of counting the number of rank-maximal matchings is  $\#P$ -complete even for strict preference lists. We then give an FPRAS for the problem by reducing it to the problem of counting the number of perfect matchings in a bipartite graph.
- In order to choose one among possibly several rank-maximal matchings in a given instance  $G$ , we consider the question of finding a rank-maximal matching that is popular among all the rank-maximal matchings in  $G$ . We call such a matching a *popular rank-maximal matching*. We show that, given a

rank-maximal matching, it can be efficiently checked whether it is a popular rank-maximal matching. If not, we output a rank-maximal matching which is more popular than the given one.

We remark that the switching graph is a weighted directed graph constructed with respect to a particular matching. In case of popular matchings, it is known from [2] that, there are at most two distinct ranked posts in an applicant’s preference list, to which he can be matched in any popular matching. This results in a switching graph with edge-weights  $\{+1, -1, 0\}$ . In case of rank-maximal matchings, the situation becomes more interesting since an applicant can be matched to one among several distinct ranked posts, and the edge-weights in the switching graph could be arbitrary. Surprisingly, the characterization still turns out to be similar to that of popular matchings, although the proofs are significantly different. We expect that the switching graph will find several applications apart from those shown in this paper.

## 2 Preliminaries

A matching  $M$  of  $G$  is a subset of edges, no two of which share an end-point. For a matched vertex  $u$ , we denote by  $M(u)$  its partner in  $M$ .

*Properties of Maximum Matchings in Bipartite Graphs.* Let  $G = (\mathcal{A} \cup \mathcal{P}, E)$  be a bipartite graph and let  $M$  be a maximum matching in  $G$ . The matching  $M$  defines a partition of the vertex set  $\mathcal{A} \cup \mathcal{P}$  into three disjoint sets, defined below:

**Definition 1 (Even, Odd, Unreachable Vertices).** *A vertex  $v \in \mathcal{A} \cup \mathcal{P}$  is even (resp. odd) if there is an even (resp. odd) length alternating path with respect to  $M$  from an unmatched vertex to  $v$ . A vertex  $v$  is unreachable if there is no alternating path from an unmatched vertex to  $v$ .*

The following lemma is well-known in matching theory; see [15] or [7] for a proof.

**Lemma 1 ([15]).** *Let  $\mathcal{E}$ ,  $\mathcal{O}$ , and  $\mathcal{U}$  be the sets of even, odd, and unreachable vertices defined by a maximum matching  $M$  in  $G$ . Then,*

- (a)  $\mathcal{E}$ ,  $\mathcal{O}$ , and  $\mathcal{U}$  are disjoint, and are the same for all the maximum matchings in  $G$ .
- (b) In any maximum matching of  $G$ , every vertex in  $\mathcal{O}$  is matched with a vertex in  $\mathcal{E}$ , and every vertex in  $\mathcal{U}$  is matched with another vertex in  $\mathcal{U}$ . The size of a maximum matching is  $|\mathcal{O}| + |\mathcal{U}|/2$ .
- (c) No maximum matching of  $G$  contains an edge with one end-point in  $\mathcal{O}$  and the other in  $\mathcal{O} \cup \mathcal{U}$ . Also,  $G$  contains no edge with one end-point in  $\mathcal{E}$  and the other in  $\mathcal{E} \cup \mathcal{U}$ .

*Rank-Maximal Matchings.* An instance of the rank-maximal matchings problem consists of a bipartite graph  $G = (\mathcal{A} \cup \mathcal{P}, E)$ , where  $\mathcal{A}$  is a set of applicants,  $\mathcal{P}$  is a set of posts, and  $E$  can be partitioned as  $E_1 \cup E_2 \cup \dots \cup E_r$ . Here  $E_i$  denotes the edges of rank  $i$ , and  $r$  denotes the maximum rank any applicant assigns to a post. An edge  $(a, p)$  has rank  $i$  if  $p$  is an  $i$ th choice of  $a$ .

**Definition 2 (Signature).** *The signature of a matching  $M$  is defined as an  $r$ -tuple  $\rho(M) = (x_1, \dots, x_r)$  where, for each  $1 \leq i \leq r$ ,  $x_i$  is the number of applicants who are matched to their  $i$ th rank post in  $M$ .*

Let  $M, M'$  be two matchings in  $G$ , with signatures  $\rho(M) = (x_1, \dots, x_r)$  and  $\rho(M') = (y_1, \dots, y_r)$ . Define  $M \succ M'$  if  $x_i = y_i$  for  $1 \leq i < k \leq r$  and  $x_k > y_k$ .

**Definition 3 (Rank-Maximal Matching).** *A matching  $M$  in  $G$  is rank-maximal if  $M$  has the maximum signature under the above ordering  $\succ$ .*

Observe that all the rank-maximal matchings in an instance have the same cardinality and the same signature.

*Computing Rank-Maximal Matchings.* Now we recall Irving et al.'s algorithm [7] for computing a rank-maximal matching in a given instance  $G = (\mathcal{A} \cup \mathcal{P}, E_1 \cup \dots \cup E_r)$ . Recall that  $E_i$  is the set of edges of rank  $i$ . For the sake of convenience, for each applicant  $a$ , we add a dummy last-resort post  $\ell(a)$  at rank  $r + 1$  in  $a$ 's preference list, and refer to the modified instance as  $G$ . This ensures that every rank-maximal matching is  $\mathcal{A}$ -complete i.e. matches all the applicants.

Let  $G_i = (\mathcal{A} \cup \mathcal{P}, E_1 \cup \dots \cup E_i)$ . The algorithm starts with  $G'_1 = G_1$  and any maximum matching  $M_1$  in  $G'_1$ .

For  $i = 1$  to  $r$  do the following and output  $M_{r+1}$ :

1. Partition the vertices in  $\mathcal{A} \cup \mathcal{P}$  into even, odd, and unreachable as in Definition 1 and call these sets  $\mathcal{E}_i, \mathcal{O}_i, \mathcal{U}_i$  respectively.
2. Delete those edges in  $E_j, j > i$ , which are incident on nodes in  $\mathcal{O}_i \cup \mathcal{U}_i$ . These are the nodes that are matched by every maximum matching in  $G'_i$ .
3. Delete all the edges from  $G'_i$  between a node in  $\mathcal{O}_i$  and a node in  $\mathcal{O}_i \cup \mathcal{U}_i$ . We refer to these edges as  $\mathcal{O}_i \mathcal{O}_i$  and  $\mathcal{O}_i \mathcal{U}_i$  edges respectively. These are the edges which do not belong to any maximum matching in  $G'_i$ .
4. Add the edges in  $E_{i+1}$  to  $G'_i$  and call the resulting graph  $G'_{i+1}$ .
5. Determine a maximum matching  $M_{i+1}$  in  $G'_{i+1}$  by augmenting  $M_i$ .

The algorithm constructs a graph  $G'_{r+1}$ . We construct a *reduced graph*  $G'$  by deleting all the edges from  $G'_{r+1}$  between a node in  $\mathcal{O}_{r+1}$  and a node in  $\mathcal{O}_{r+1} \cup \mathcal{U}_{r+1}$ . The graph  $G'$  will be used in subsequent sections.

We note the following invariants of Irving et al.'s algorithm:

(I1) For every  $1 \leq i \leq r$ , every rank-maximal matching in  $G_i$  is contained in  $G'_i$ .



- (I2) The matching  $M_i$  is rank-maximal in  $G_i$ , and is a maximum matching in  $G'_i$ .
- (I3) If a rank-maximal matching in  $G$  has signature  $(s_1, \dots, s_i, \dots, s_r)$  then  $M_i$  has signature  $(s_1, \dots, s_i)$ .
- (I4) The graphs  $G'_i$ ,  $1 \leq i \leq r + 1$  constructed at the end of iteration  $i$  of Irving et al.'s algorithm, and  $G'$  are independent of the rank-maximal matching computed by the algorithm. This follows from Lemma 1 and invariant I2.

### 3 Switching Graph Characterization

In this section, we describe the switching graph characterization of rank-maximal matchings, and show its application in computing *rank-maximal pairs*.

Let  $M$  be a rank-maximal matching in  $G$  and let  $G' = (\mathcal{A} \cup \mathcal{P}, E')$  be the reduced graph as described in Section 2.

**Definition 4 (Switching Graph).** *The switching graph  $G_M = (V_M, E_M)$  with respect to a rank-maximal matching  $M$  is a directed weighted graph with  $V_M = \mathcal{P}$  and  $E_M = \{(p_i, p_j) \mid \exists a \in \mathcal{A}, (a, p_i) \in M, (a, p_j) \in E'\}$ . Further, weight of an edge  $(p_i, p_j)$  is  $w(p_i, p_j) = \text{rank}(a, p_j) - \text{rank}(a, p_i)$ , where  $\text{rank}(a, p)$  is the rank of a post  $p$  in the preference list of an applicant  $a$ .*

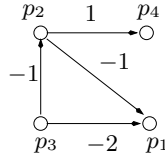
Thus an edge  $(p_i, p_j) \in E_M$  iff there exists an applicant  $a$  such that  $(a, p_i) \in M$  and  $(a, p_j)$  is an edge in the graph  $G'$ . We define the following notation:

1. *Sink vertex:* A vertex  $p$  of  $G_M$  is called a *sink* vertex, if  $p$  has no outgoing edge in  $G_M$  and  $p \in \mathcal{E}_1 \cap \mathcal{E}_2 \cap \dots \cap \mathcal{E}_{r+1}$ . Recall that  $\mathcal{E}_i$  is the set of vertices which were even in the graph  $G'_i$  constructed in the  $i$ th iteration of Irving et al.'s algorithm.
2. *Sink and non-sink components of  $G_M$ :* A connected component  $\mathcal{X}$  in the underlying undirected graph of  $G_M$  is called a *sink component* if  $\mathcal{X}$  contains one or more sink vertices, and a *non-sink component* otherwise.
3. *Switching paths and switching cycles:* A path  $T = \langle p_0, p_1 \dots, p_{k-1} \rangle$  in  $G_M$  is called a *switching path* if  $T$  ends in a sink vertex and  $w(T) = 0$ . Here,  $w(T)$  is the sum of the weights of the edges in  $T$ . A cycle  $C = \langle p_0, \dots, p_{k-1}, p_0 \rangle$  in  $G_M$  is called a *switching cycle* if  $w(C) = 0$ .
4. *Switching operation:* Let  $T = \langle p_0, p_1 \dots, p_{k-1} \rangle$  be a switching path in  $G_M$ . Let  $\mathcal{A}_T = \{a \in \mathcal{A} \mid M(a) \in T\}$ . Further, let  $M(a_i) = p_i$  for  $0 \leq i \leq k - 2$ . We denote by  $M' = M \cdot T$ , the matching obtained by *applying*  $T$  to  $M$ . Thus, for  $a_i \in \mathcal{A}_T$ ,  $M'(a_i) = p_{i+1}$ , and for  $a \notin \mathcal{A}_T$ ,  $M'(a) = M(a)$ . The matching  $M \cdot C$ , obtained by *applying* a switching cycle  $C$  to  $M$  is defined analogously. We also refer to  $M \cdot C$  or  $M \cdot T$  as a *switching operation*.

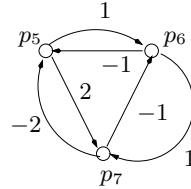
Figure 1 illustrates an example instance along with its switching graph.

- $a_1 : p_1 \ p_2 \ p_3$
- $a_2 : p_1 \ p_2 \ p_4$
- $a_3 : p_1$
- $a_4 : p_5 \ p_6 \ p_7$
- $a_5 : p_5 \ p_6 \ p_7$
- $a_6 : p_5 \ p_6 \ p_7$

(a)



(b)



**Fig. 1.** (a) Preference lists of agents  $\{a_1, \dots, a_6\}$  in increasing order of ranks. (b) Switching graph  $G_M$  with respect to rank-maximal matching  $M = \{(a_1, p_3), (a_2, p_2), (a_3, p_1), (a_4, p_7), (a_5, p_5), (a_6, p_6)\}$ . The vertex  $p_4$  is the only sink-vertex and the path  $(p_3, p_2, p_4)$  is a switching path. Note that every directed cycle is a switching cycle.

### 3.1 Properties of the Switching Graph

In this section, we prove several useful properties of the switching graph by characterizing switching paths and switching cycles.

In the following lemma, we show that a switching operation on a rank-maximal matching  $M$  results in another rank-maximal matching in  $G$ .

**Lemma 2.** *Let  $T$  (resp.  $C$ ) be a switching path (resp. switching cycle) in  $G_M$ . Then,  $M' = M \cdot T$  (resp.  $M' = M \cdot C$ ) is a rank-maximal matching in  $G$ .*

*Proof.* We prove the lemma for a switching path  $T$ . A similar argument follows for a switching cycle. To show that  $M'$  is rank-maximal, we show that  $M$  and  $M'$  have the same signature.

Let  $T = \langle p_0, p_1, \dots, p_{k-1} \rangle$  be a switching path in  $G_M$ . Let  $\mathcal{A}_T = \{a \mid M(a) \in T\}$ . By the definition of a *switch*, we know that  $|M| = |M'|$  and for each  $a \notin \mathcal{A}_T$ , we have  $M'(a) = M(a)$ . Thus, it suffices to show that the signatures of  $M$  and  $M'$  restricted to the applicants in  $\mathcal{A}_T$  are the same. We denote them by  $\rho_T(M) = (x_1, x_2, \dots, x_r)$  and  $\rho_T(M') = (y_1, y_2, \dots, y_r)$  respectively. Note that an edge of rank  $i$  in  $M$  contributes  $-i$  to the weight of  $T$ , whereas one in  $M'$  contributes  $i$ . Further, since  $T$  is a switching path,  $w(T) = 0$ . Thus,

$$w(T) = (y_1 - x_1) + 2(y_2 - x_2) + \dots + r(y_r - x_r) = 0 \tag{1}$$

Since we consider only applicants in  $\mathcal{A}_T$ , we know that,  $\sum_{i=1}^r x_i = \sum_{i=1}^r y_i$ , i.e.,

$$\sum_{i=1}^r (x_i - y_i) = 0 \tag{2}$$

For contradiction, assume that  $\rho_T(M) \succ \rho_T(M')$ . That is, there exists an index  $j$  such that  $x_j > y_j$  and, for  $1 \leq i < j$ , we have  $x_i = y_i$ . Then, for Eqn. 2 to be satisfied, there exists an index  $\ell > j$  such that  $x_\ell < y_\ell$ . In fact we will show the following stronger claim:

*Claim.* There exists an index  $\ell > j$  such that  $\sum_{i=1}^{\ell} (x_i - y_i) < 0$ .

Before proving the claim, we show how it suffices to complete the proof of the lemma. Assuming the claim, consider the reduced graph  $G'_\ell$  constructed in the  $\ell$ th iteration of Irving et al.'s algorithm.

As  $\sum_{i=1}^{\ell} (x_i - y_i) < 0$ , we have  $\sum_{i=1}^{\ell} x_i < \sum_{i=1}^{\ell} y_i$ . Thus  $|M \cap G'_\ell| < |M' \cap G'_\ell|$ . However, by Invariant (I2) (ref. Section 2), this contradicts the fact that every rank-maximal matching restricted to any rank  $\ell$  is also a maximum matching in the reduced graph  $G'_\ell$ . This completes the proof of the lemma. We prove the claim below.

*Proof (of claim):* Assume the contrary, i.e.  $\sum_{i=1}^k (x_i - y_i) \geq 0$  for all  $k$ . Note that this is trivially true for  $k \leq j$ , by our choice of  $j$ . Equivalently,  $\sum_{i=k+1}^r (x_i - y_i) \leq 0$  for all  $k$ . Define  $T_k = \sum_{i=k}^r (x_i - y_i)$  for  $1 \leq k \leq r$ . Thus, to prove the claim, it suffices to show that there exists an index  $\ell$  such that  $T_\ell > 0$ . Now consider Eqn. 1. It can be rewritten as follows:

$$(x_1 - y_1) + 2(x_2 - y_2) + \dots + r(x_r - y_r) = T_1 + T_2 + \dots + T_r = 0 \quad (3)$$

We know that  $T_1 = 0$ , because it is the left-side of Eqn. 2. Now, consider the term  $T_r = x_r - y_r$ . If  $T_r = 0$ , we can eliminate  $x_r$  and  $y_r$  and get equations in  $r - 1$  variables. If  $T_r > 0$ , then Eqn. 2 implies that the claim holds for  $k = r - 1$ . So, without loss of generality, we can assume  $T_r < 0$ . But then, to satisfy Eqn. 3, there exists an index  $i$ ,  $1 < i < r$ , such that  $T_i > 0$ . This implies that the claim holds for  $\ell = i - 1$ . This completes the proof of the claim.  $\square$

Now we address the question of recognition of switching paths and switching cycles in  $G_M$ . In Lemma 3, we show that every cycle in  $G_M$  is in fact a switching cycle, that is, a zero-weight cycle. In Lemma 4, we characterize switching paths. The proofs for both these lemmas can be found in [4].

**Lemma 3.** *Let  $M$  be a rmm in  $G$ , and  $C$  be a cycle in  $G_M$ . Then  $w(C) = 0$ .*

*Proof.* (Sketch) Let  $C'$  be the alternating cycle in  $G'$ , corresponding to the cycle  $C$  in  $G_M$ . To prove the Lemma, it suffices to show that,  $C'$  has an equal number of matched and unmatched edges of any rank  $i$ , and hence  $w(C) = 0$ . We complete the proof by induction on  $i$ .  $\square$

**Lemma 4.** *Let  $M$  be a rmm in  $G$ , and  $G_M$  be the switching graph with respect to  $M$ . Recall that  $\mathcal{E}_i$  is the set of even vertices in the graph  $G'_i$  constructed in the  $i$ th iteration of Irving et al.'s algorithm. The following properties hold :*

1. *Let  $p$  be an unmatched post in  $M$ . Then  $p \in \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r+1}$  and therefore is a sink in  $G_M$ .*
2. *A post  $p$  belongs to a sink component iff  $p \in \mathcal{E}_{r+1}$ . A post  $p$  belongs to a non-sink component iff  $p \in \mathcal{U}_{r+1}$ .*
3. *Let  $T$  be a path from a post  $p$  to some sink  $q$  in  $G_M$ . Then  $w(T) = 0$  iff  $p \in \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r+1}$ .*

In the following theorem, we prove that every rank-maximal matching can be obtained from  $M$  by applying suitable *switches*.

**Theorem 1.** *Every rank-maximal matching  $M'$  in  $G$  can be obtained from  $M$  by applying to  $M$  vertex-disjoint switching paths and switching cycles in  $G_M$ .*

*Proof.* Consider any rank-maximal matching  $M'$  in  $G$ . We show that  $M'$  can be obtained from  $M$  by applying a set of vertex-disjoint switching paths and switching cycles of  $G_M$ . Consider  $M \oplus M'$  which is a collection of vertex-disjoint paths and cycles in  $G$ . Also note that the cycles and paths contain alternating edges of  $M$  and  $M'$ . We show that the paths and cycles in  $M \oplus M'$  are switching paths and switching cycles in  $G_M$ .

From the invariants of Irving et al.'s algorithm mentioned in Section 2, all the edges of  $M$  and  $M'$  are also present in  $G'$ . A cycle in  $M \oplus M'$  has alternating edges of  $M$  and  $M'$ , and hence has a corresponding directed cycle in  $G_M$ . As proved in Lemma 3, every cycle in  $G_M$  is a switching cycle.

Now we consider paths in  $M \oplus M'$ . All the paths are of even length, since all the rank-maximal matchings are of the same cardinality. Let  $T_G = \langle p_1, a_1, \dots, p_k, a_k, p_{k+1} \rangle$  be any even-length path in  $M \oplus M'$  with  $p_{k+1}$  unmatched in  $M$  and  $p_1$  unmatched in  $M'$ . For every  $1 \leq i \leq k$ , let  $M(p_i) = a_i$ . It is easy to see that the path  $T = \langle p = p_1, p_2, \dots, p_{k+1} = p' \rangle$  is present in  $G_M$  and it ends in a sink  $p'$ . Our goal is to show that  $w(T) = 0$ . For this, we prove that  $p_1 \in \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r+1}$ . Note that  $M'$  is a rank-maximal matching in  $G$  and  $M'$  leaves the post  $p = p_1$  unmatched. As every post in  $\mathcal{O}_i \cup \mathcal{U}_i$  for any  $i$  is matched in every rank-maximal matching,  $p_1 \notin \mathcal{O}_i \cup \mathcal{U}_i$  for  $1 \leq i \leq r + 1$ . Therefore  $p_1 \in \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r+1}$ ; Thus, using Lemma 4, we can conclude that the path  $T$  has weight  $w(T) = 0$  in  $G_M$ , and hence is a switching path in  $G_M$ .

Applying these switching paths and cycles to  $M$  gives us the desired matching  $M'$ , thus completing the proof.  $\square$

### 3.2 Generating All Rank-Maximal Pairs

In this section we give an efficient algorithm to compute the set of rank-maximal pairs, defined below:

**Definition 5.** *An edge  $(a, p)$  is a rank-maximal pair if there exists a rank-maximal matching  $M$  in  $G$  such that  $M(a) = p$ .*

We refer to rank-maximal pairs as *rmm-pairs*. We show that the set of rmm-pairs can be computed in time linear in the size of the switching graph  $G_M$  constructed with respect to any rank-maximal matching  $M$ . We prove the following theorem:

**Theorem 2.** *The set of rmm-pairs for an instance  $G = (\mathcal{A} \cup \mathcal{P}, E)$  can be computed in  $O(\min(n + r, r\sqrt{n})m)$  time.*

*Proof.* We note that, by Theorem 1, an edge  $(a, p)$  is a rmm-pair iff (i)  $(a, p) \in M$  or, (ii) the edge  $(M(a), p)$  belongs to a switching cycle in  $G_M$  or, (iii) the edge  $(M(a), p)$  belongs to a switching path in  $G_M$ .

Condition (i) can be checked by computing a rank-maximal matching  $M$  which takes  $O(\min(n + r, r\sqrt{n})m)$  time. Condition (ii) can be checked by computing strongly connected components of  $G_M$ , which takes time linear in the size of  $G_M$ .

To check Condition (iii), note that a post  $p$  has a zero-weight path to a sink if and only if  $p \in \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r+1}$  by Lemma 4 (3). Moreover, all the paths from such a post  $p$  to a sink have weight zero. Therefore, performing a DFS from each  $p \in \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r+1}$  and marking all the edges encountered in the DFS (not just the tree edges) gives all the pairs which satisfy Condition (iii).  $\square$

## 4 Counting Rank-Maximal Matchings

We prove that the problem of counting the number of rank-maximal matchings in an instance is  $\#P$ -complete, and give an FPRAS for the same.

### 4.1 Hardness of Counting

We prove  $\#P$ -hardness by reducing the problem of counting the number of matchings in 3-regular bipartite graphs to counting the number of rank-maximal matchings. The former was shown to be  $\#P$ -complete by Dagum and Luby [3].

**Reduction for lists with ties:** First let us consider the case when preference lists may contain ties<sup>1</sup>. Let  $H = (X \cup Y, E)$  be a 3-regular bipartite graph. We construct an instance  $G$  of the rank-maximal matchings problem by setting  $G = H$  and assigning rank 1 to all the edges in  $E$ . It is well-known that a  $k$ -regular bipartite graph admits a perfect matching for any  $k$ . It is easy to see that every perfect matching in  $H$  is a rank-maximal matching in  $G$  and vice versa. This proves the  $\#P$ -hardness of the problem for the case of ties.

**Reduction for strict lists:** Let  $H = (X \cup Y, E)$  be a 3-regular bipartite graph, with  $|X| = |Y| = n$ . The corresponding instance  $G = (\mathcal{A} \cup \mathcal{P}, E_G)$  of the rank-maximal matchings problem is as follows:

$$\mathcal{A} = \{a_x : x \in X\} \cup \{ad_1, ad_2, \dots, ad_{n-3}\}; \mathcal{P} = \{p_y : y \in Y\} \cup \{pd_1, pd_2, \dots, pd_{n-3}\}$$

Here  $ad_i, pd_i, 1 \leq i \leq n - 3$  are dummy agents and dummy posts respectively.

To construct the preference lists of agents in  $\mathcal{A}$ , we fix an arbitrary ordering on the vertices in  $Y$  i.e.  $order : Y \rightarrow \{1, \dots, n\}$ . This assigns an ordering on the posts in  $\mathcal{P}$ . The preference lists of the agents can be described as below:

- A dummy agent  $ad_i$  has a preference list of length one, with dummy post  $pd_i$  as his rank 1 post.
- The preference list of an agent  $a_x$  consists of posts  $p_{y_1}, p_{y_2}, p_{y_3}$  ranked at  $order(y_1), order(y_2)$ , and  $order(y_3)$  respectively, where  $y_1, y_2, y_3$  denote the 3 neighbors of  $x$  in  $H$ . The remaining places in the preference list of  $a_x$  are filled using the  $n - 3$  dummy posts.

---

<sup>1</sup> Recall that preference lists are said to contain ties if an applicant ranks two or more posts at the same rank.

Following Lemma (see [4] for proof) shows the correctness of the reduction.

**Lemma 5.** *Let  $H$  be a 3-regular bipartite graph and let  $G$  be the rank-maximal matchings instance constructed from  $H$  as above. There is a one-to-one correspondence between perfect matchings in  $H$  and rank-maximal matchings in  $G$ .*

Using Lemma 5 and our observation for ties, we conclude the following:

**Theorem 3.** *The problem of counting the number of rank-maximal matchings in an instance is #P-Complete for both strict and tied preference lists.*

### 4.2 An FPRAS for Counting Rank-Maximal Matchings

Given the hardness result in Section 4.1, it is unlikely to be able to count the number of rank-maximal matchings in an instance in polynomial time. We now show that there exists a fully polynomial-time randomized approximation scheme (FPRAS) for the problem. We use the following result by Jerrum et al. [9]:

**Theorem 4 ([9]).** *There exists an FPRAS for the problem of counting the number of perfect matchings in a bipartite graph.*

We give a polynomial-time reduction from the problem of counting the number of rank-maximal matchings (denoted as #RMM) to the problem of counting the number of perfect matchings in a bipartite graph (denoted as #BPM).

**Reduction from #RMM to #BPM:** Given an instance  $G = (\mathcal{A} \cup \mathcal{P}, E)$  of the rank-maximal matchings problem, we first construct another instance  $H$  of the rank-maximal matchings problem, which is used to get an instance  $I$  of the bipartite perfect matchings problem. The steps of the construction are as follows:

1. For every  $a \in \mathcal{A}$ , introduce a dummy last-resort post  $\ell(a)$  ranked  $r + 1$ . This ensures that every rank-maximal matching is  $\mathcal{A}$ -complete.
2. Let  $M$  be any rank-maximal matching in  $G$ , let  $G'$  be the reduced graph obtained by Irving et al.'s algorithm (ref. Section 2).
3. Let  $k$  be the number of unmatched posts in  $G'$ . Introduce  $k$  dummy applicants  $ad_1, \dots, ad_k$ . The preference list of each dummy applicant consists of all the posts in  $G'$  which are in  $\mathcal{E}_1 \cap \dots \cap \mathcal{E}_{r+1}$ , tied at rank  $r + 2$ .
4. The instance  $H$  consists of all the applicants in  $G$  and their preference lists in  $G$ , together with the dummy applicants and their preference lists introduced above. The set of posts in  $H$  is the same as that in  $G$ .
5. The instance  $I$  of bipartite perfect matchings problem is simply the reduced graph  $H'$ , obtained by executing Irving et al.'s algorithm on  $H$ .

Correctness of the reduction follows from the following lemma, the proof (in [4]) uses the switching graph characterization.

**Lemma 6.** *Let  $G$  be the rank-maximal matchings instance and let  $H$  and  $I$  be the rank-maximal matchings instance and the bipartite perfect matchings instance respectively constructed as above. Then, the following hold:*

1. Corresponding to each rank-maximal matching  $M$  in  $G$ , there are exactly  $k!$  distinct rank-maximal matchings in  $H$ .
2. Each rank-maximal matching in  $H$  matches all the applicants and posts, and all its edges appear in  $I$ . Hence it is a perfect matching in the instance  $I$ .
3. A matching in  $G$  that is not rank-maximal has no corresponding perfect matching in  $I$ .

The FPRAS for #RMM involves the following steps:

1. The reduction from #RMM instance  $G$  to #BPM instance  $I$ ,
2. Running Jerrum et al.'s FPRAS on  $I$  to get an approximate count, say  $C$ , of the number of perfect matchings in  $I$ ,
3. Dividing  $C$  by  $k!$  to get an approximate count of number of rank-maximal matchings in  $G$ .

Steps 1 and 2 clearly work in polynomial time. For step 3, note that both  $C$  and  $k$  are at most  $n!$  and can be represented in  $O(n \log n)$  bits, which is polynomial in the size of  $G$ . Therefore Step 3 also works in polynomial time. This completes the FPRAS for #RMM problem.

## 5 Popularity of Rank-Maximal Matchings

As mentioned earlier, an instance of the rank-maximal matchings problem may admit more than one rank-maximal matching. To choose one rank-maximal matching, it is natural to impose an additional optimality criterion. Such a question has been considered earlier in the context of popular matchings by [10, 12] and also in the context of the stable marriage problem [8]. The additional notion of optimality that we impose is the notion of popularity, defined below:

**Definition 6 (Popular Matching).** *A matching  $M$  is more popular than matching  $M'$  (denoted by  $M \succ_p M'$ ) if the number of applicants that prefer  $M$  to  $M'$  is more than the number of applicants that prefer  $M'$  to  $M$ . A matching  $M$  is popular if no matching  $M'$  is more popular than  $M$ .*

An applicant  $a$  prefers matching  $M$  to  $M'$  if either (i)  $a$  is matched in  $M$  and unmatched in  $M'$ , or (ii)  $a$  is matched in both and prefers the post  $M(a)$  to  $M'(a)$ . We consider the following question: Given an instance of the rank-maximal matchings problem, is there a rank-maximal matching that is popular in the set of all rank-maximal matchings? We refer to such a matching as a *popular rank-maximal matching*. There are simple instances in which there is no popular matching; further there is no popular rank-maximal matching. However, if a popular rank-maximal matching exists, it seems an appealing choice since it enjoys both rank-maximality and popularity. We make partial progress on this question. Using the switching graph characterization developed in Section 3, we give a simple algorithm to determine if a given rank-maximal matching  $M$  is a popular rank-maximal matching. If not, our algorithm outputs a rank-maximal matching  $M'$  which is more popular than  $M$ .

*Outline of the Algorithm.* Given a graph  $G = (\mathcal{A} \cup \mathcal{P}, E)$  and a rank-maximal matching  $M$  in  $G$ , the algorithm first constructs the switching graph  $G_M$  corresponding to  $M$ . Now consider the following re-weighted graph  $\tilde{G}_M$  where positive weights of edges in  $G_M$  are replaced by  $+1$  weights and negative weights by  $-1$ . Thus a  $-1$  weight edge  $(p_i, p_j)$  in  $\tilde{G}_M$  implies that  $M(p_i)$  prefers  $p_j$  to  $p_i$ .

Let  $T$  be a switching path in  $G_M$ , and let  $\tilde{T}$  be the corresponding path in  $\tilde{G}_M$ . It is easy to see that if  $w(\tilde{T}) < 0$  in  $\tilde{G}_M$ , then  $M' = M \cdot T$  is more popular than  $M$ . Same holds for a switching cycle in  $G_M$ . Therefore,  $M$  is a popular rank-maximal matching, if and only if there is no negative-weight path to sink or negative-weight cycle in  $\tilde{G}_M$ .

To check this, we use shortest path computations using Bellman-Ford algorithm in a suitably modified graph. The details of the algorithm and proof of the following lemma, which establishes correctness, can be found in [4].

**Lemma 7.** *A given rank-maximal matching  $M$  is popular if and only if there is no negative-weight path to a sink or a negative-weight cycle in the re-weighted switching graph.*

Thus we get an  $O(mn)$  time algorithm for checking whether a given rank-maximal matching is a popular rank-maximal matching, where  $m$  and  $n$  are number of edges and vertices in the switching graph respectively.

**Acknowledgments.** We thank Partha Mukhopadhyay for a proof of Lemma 2.

## References

1. Abraham, D.J., Cechlárová, K., Manlove, D.F., Mehlhorn, K.: Pareto optimality in house allocation problems. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 3–15. Springer, Heidelberg (2004)
2. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. *SIAM Journal on Computing* **37**(4), 1030–1045 (2007)
3. Dagum, P., Luby, M.: Approximating the permanent of graphs with large factors. *Theor. Comput. Sci.* **102**(2), 283–305 (1992)
4. Ghoshal, P., Nasre, M., Nimbhorkar, P.: Rank maximal matchings - structure and algorithms. CoRR, abs/1409.4977 (2014)
5. Hylland, A., Zeckhauser, R.: The efficient allocation of individuals to positions. *Journal of Political Economy* **87**(2), 293–314 (1979)
6. Irving, R.W.: Greedy matchings. Technical Report, University of Glasgow, TR-2003-136 (2003)
7. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.E.: Rank-maximal matchings. *ACM Transactions on Algorithms* **2**(4), 602–610 (2006)
8. Irving, R.W., Leather, P., Gusfield, D.: An efficient algorithm for the “optimal” stable marriage. *Journal of the ACM* **34**(3), 532–543 (1987)
9. Jerrum, M., Sinclair, A., Vigoda, E.: A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM* **51**(4), 671–697 (2004)
10. Kavitha, T., Nasre, M.: Note: Optimal popular matchings. *Discrete Applied Mathematics* **157**(14), 3181–3186 (2009)



11. Kavitha, T., Shah, C.D.: Efficient algorithms for weighted rank-maximal matchings and related problems. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 153–162. Springer, Heidelberg (2006)
12. McDermid, E., Irving, R.W.: Popular matchings: structure and algorithms. *Journal of Combinatorial Optimization* **22**(3), 339–358 (2011)
13. Nasre, M.: Popular Matchings: Structure and Cheating Strategies. In: Proceedings of 30th STACS, pp. 412–423 (2013)
14. Paluch, K.: Capacitated rank-maximal matchings. In: Spirakis, P.G., Serna, M. (eds.) CIAC 2013. LNCS, vol. 7878, pp. 324–335. Springer, Heidelberg (2013)
15. Pulleyblank, W.R.: Matchings and Extensions. *Handbook of Combinatorics*, vol. 1, pp. 179–232. MIT Press, Cambridge (1995)
16. Yuan, Y.: Residence exchange wanted: A stable residence exchange problem. *European Journal of Operational Research* **90**(3), 536–546 (1996)

# The Generalized Popular Condensation Problem

Yen-Wei Wu<sup>1</sup>, Wei-Yin Lin<sup>1</sup>, Hung-Lung Wang<sup>4</sup>, and Kun-Mao Chao<sup>1,2,3</sup> (✉)

<sup>1</sup> Department of Computer Science and Information Engineering,  
National Taiwan University, Taipei 106, Taiwan

<sup>2</sup> Graduate Institute of Biomedical Electronics and Bioinformatics,  
National Taiwan University, Taipei 106, Taiwan

<sup>3</sup> Graduate Institute of Networking and Multimedia,  
National Taiwan University, Taipei 106, Taiwan  
kmchao@csie.ntu.edu.tw

<sup>4</sup> Institute of Information and Decision Sciences,  
National Taipei University of Business, Taipei 100, Taiwan

**Abstract.** In this paper, we are concerned with the *generalized popular condensation problem*, which is an extension of the popular matching problem. An instance of the popular matching problem consists of a set of applicants  $A$ , a set of posts  $P$ , and a set of preference lists. According to the preference lists, the goal is to match each applicant with at most one unique post so that the resulting matching is “popular.” A matching  $M$  mapping from  $A$  to  $P$  is *popular* if there is no other matching  $M'$  such that more applicants prefer  $M'$  to  $M$  than prefer  $M$  to  $M'$ . However, such a matching does not always exist. To fulfill the popular matching requirements, a possible manipulation is to neglect some applicants. Given that each applicant is appended with a cost for neglecting, the generalized popular condensation problem asks for a subset of applicants, with minimum sum of costs, whose deletion results in an instance admitting a popular matching. We prove that this problem is NP-hard, and it is also NP-hard to approximate to within a certain factor, assuming ties are involved in the preference lists. For the special case where the costs of all applicants are equal, we show that the problem can be solved in  $O(\sqrt{nm})$  time, where  $n$  is the number of applicants and posts, and  $m$  is the total length of the preference lists.

## 1 Introduction

The popular matching problem has been proposed for decades [4], and recently characteristic and algorithmic results were given by Abraham *et al.* [1]. Later, plenty of extensions or variants emerged for different criteria [2, 9, 15–17, 19]. In the popular matching problem, one is given a set of applicants  $A$ , a set of posts  $P$ , and a set of preference lists that correspond one-to-one to the applicants in  $A$ . Each preference list is a sequence of posts, representing the preference of posts given by a specific applicant. A popular matching  $M$  defined as follows is a way of assigning at most one post for each applicant, and no two applicants get the same post. We say that an applicant  $a$  *prefers* a matching  $M$  to  $M'$  if

$a$  is matched to a post  $p$  in  $M$  and a post  $p'$  in  $M'$ , where  $a$  prefers  $p$  to  $p'$ , or  $a$  is matched in  $M$  but not matched in  $M'$ . A matching  $M$  is *popular* if there is no other matching  $M'$  such that more applicants prefer  $M'$  to  $M$  than prefer  $M$  to  $M'$ . Abraham *et al.* [1] gave an  $O(\sqrt{nm})$ -time algorithm to find a popular matching if it exists, where  $n$  is the number of applicants and posts, and  $m$  is the total length of the preference lists.

Popular matchings for a given instance do not always exist. For example, suppose there are three applicants and three posts  $p_1, p_2, p_3$  and each applicant ranks  $p_1, p_2, p_3$  in exactly the same way. It is not difficult to verify that there exists no popular matching in such an instance. For those instances which do not have a popular matching, there are many investigations focusing on this matter. Manipulations such as adding more posts [12] or neglecting some applicants [22] so that the resulting instance admits a popular matching are proposed.

The manipulation of neglecting some applicants is modeled as the *popular condensation problem* [22]. In that problem, one is asked for the least number of applicants to be neglected such that there is a popular matching in the resulting instance. In this paper, we generalize the popular condensation problem in such a way that all applicants are appended with costs, and the goal is to find a subset of applicants, with minimum sum of costs, to be neglected such that the resulting instance admits a popular matching.

## 1.1 Related Work

Since the popular matching problem was proposed by Gardenfors [4] and was characterized by Abraham *et al.* [1] in 2005, there have been several exciting extensions in recent years. For those instances with more than one popular matching, Kavitha and Nasre [11] and McDermid and Irving [18] investigated the notions of optimality among all popular matchings. There are also extensions such as weighted popular matchings [19], many-to-one [16] and many-to-many matchings [20], dynamic matchings (where applicants and posts can be inserted or deleted dynamically) [2], random popular matchings [15], and popular matchings in the stable marriage and the roommates problems [3, 6, 7, 10].

Similar to our motivation, there are some investigations that focus on the case where no popular matching exists in the instance. The popular condensation problem proposed by Wu *et al.* [22] focuses on the case where preference lists are strictly ordered. The authors proposed an  $O(n+m)$ -time algorithm. In contrast to the idea of condensing the size of the set of applicants, Kavitha and Nasre [12] found another substitute solution by expanding the set of posts until the instance admits a popular matching. In other words, they asked for the minimum number of posts to copy for an instance to have a popular matching, where each post is assigned with an upper bound on the number of copies. They proved that this problem is NP-hard. The authors also gave a polynomial-time algorithm for a variant of the problem above, where the total number of copies is bounded by an integer. In [13], Kavitha *et al.* considered the problem of augmenting  $G$  at minimum cost such that the new instance admits a popular matching, given that each item (namely, post) is associated with a copy cost. They proved that

this problem is NP-hard. Furthermore, it is NP-hard to approximate to within a factor of  $\sqrt{n_1}/2$ , where  $n_1$  is the number of applicants.

### 1.2 Problem Definition

In this subsection, we formally define the generalized popular condensation problem. An instance of this problem consists of

- a set of *applicants*, denoted by  $A$ ;
- a set of *posts*, denoted by  $P$ ;
- a set of *preference lists*, each of which ranks a subset of posts for an applicant;
- a *cost* function  $c: A \rightarrow \mathbb{R}^+ \cup \{0\}$ .

The instance is often formulated as a bipartite graph  $G = (A \cup P, E)$ , called the *instance graph*, where  $(a, p) \in E$  if  $p$  is on  $a$ 's preference list. We denote by  $r_G(a, p)$  the rank of post  $p$  on  $a$ 's preference list. Note that there may be multiple posts of the same rank on an applicant's preference list. We say that  $a$  *prefers*  $p$  to  $p'$  if  $r_G(a, p) < r_G(a, p')$  and denote such a relation by  $p \prec_a p'$ . If  $r_G(a, p) = r_G(a, p')$ , the two posts  $p$  and  $p'$  are *indifferent* to  $a$ , and we denote the relation by  $p \simeq_a p'$ . For simplicity, we omit the subscript if there is no ambiguity. For any two matchings  $M$  and  $M'$  of  $G$ , we say applicant  $a$  *prefers*  $M$  to  $M'$  if  $r_G(a, p) < r_G(a, p')$  where  $(a, p) \in M$  and  $(a, p') \in M'$ , or  $a$  is matched in  $M$  but not in  $M'$ .

Given an instance graph  $G$ , a matching  $M$  is said to be *popular* if there exists no matching  $M'$  such that more applicants prefer  $M'$  to  $M$  than prefer  $M$  to  $M'$ . As already observed, popular matchings do not always exist. We say that an instance  $H = (A_H \cup P, E_H)$  is a *popular condensation* of a given instance graph  $G = (A \cup P, E)$  if  $H$  admits a popular matching,  $A_H \subseteq A$ ,  $E_H = \{(a, p) \in E : a \in A_H\}$ , and  $r_H(a, p) = r_G(a, p)$  for all  $(a, p) \in E_H$ . The set  $A \setminus A_H$  is called a *condensing set* of  $G$  with respect to  $H$ , and the *cost* of  $H$  is the sum of the costs of those applicants in its condensing set, i.e.,  $\sum_{a \in A \setminus A_H} c(a)$ . Let  $\text{Cond}(G)$  denote the set of all popular condensations of  $G$ . The following is a formal definition of the generalized popular condensation problem.

*Problem 1 (The Generalized Popular Condensation Problem).* Given an instance graph  $G = (A \cup P, E)$ , the generalized popular condensation problem asks for a condensing set  $D^*$  of  $G$  with respect to  $H^*$ , where  $H^*$  is a popular condensation  $H^* = (A_{H^*} \cup P, E_{H^*})$  of  $G$  satisfying

$$H^* = \arg \min_{H \in \text{Cond}(G)} \sum_{a \in A \setminus A_H} c(a).$$

For an instance graph  $G$ , a condensing set  $D^*$  we seek in Problem 1 is called an *optimal condensing set*, and a popular condensation  $H^*$  is called an *optimal popular condensation*. Clearly, for any instance  $G = (A \cup P, E)$ , if  $|A| - 1$  applicants are neglected, then the resulting instance must admit a popular matching. Therefore, a popular condensation sought in Problem 1 always exists.

The rest of this paper is organized as follows. First, we highlight some essential properties of popular matchings in Section 2. Then, we prove the NP-hardness of the generalized popular condensation problem in Section 3, and show the inapproximability of this problem. For the special case where all the costs are the same, we show that the problem can be solved in  $O(\sqrt{nm})$  time in Section 4, where  $n$  is the number of applicants and posts, and  $m$  is the total length of the preference lists. Finally, we summarize our main results in Section 5.

## 2 Preliminaries

In this section, we define the notation and summarize some previous results which are related to our work. We first briefly introduce the idea of the Gallai-Edmonds decomposition on the vertex set of a bipartite graph [5]. Given a bipartite graph  $G = (A \cup P, E)$  and a matching  $M$  of  $G$ , an  $M$ -alternating path is a path on which any pair of adjacent edges contains exactly one edge in  $M$ . Suppose that  $M$  is a maximum matching of  $G$ . A node  $v$  is *even* (respectively *odd*) if there is an  $M$ -alternating path of even (respectively odd) length leading from an unmatched vertex to  $v$ . A node  $v$  is *unreachable* if there is no  $M$ -alternating path leading from an unmatched vertex to  $v$ . Let  $\mathcal{E}_G, \mathcal{O}_G$ , and  $\mathcal{U}_G$  denote the set of all even, odd, and unreachable vertices, respectively. Clearly,  $\{\mathcal{E}_G, \mathcal{O}_G, \mathcal{U}_G\}$  is a partition of  $A \cup P$ . Such a partition is called the Gallai-Edmonds decomposition. For any edge  $(a, p) \in E$ , we say that  $(a, p)$  is of type  $X$ - $Y$  if  $a \in X$  and  $p \in Y$ , where  $X, Y \in \{\mathcal{E}_G, \mathcal{O}_G, \mathcal{U}_G\}$ .

For an instance graph  $G = (A \cup P, E)$ , if  $p \in P$  is the top-ranked post on some applicant's preference list, we say that  $p$  is an  $f$ -post with respect to  $G$  and denote the set of all  $f$ -posts of  $G$  by  $F_G$ . Let  $f_G(a)$  denote the set of top-ranked posts on  $a$ 's preference list. The *first-choice graph* of  $G$  is defined as  $G_1 = (A \cup P, E_1)$ , where  $E_1 = \{(a, p) \in E: r_G(a, p) = 1\}$ , that is, the graph consists of all rank-one edges. As in [1], there is a unique last resort post appended at the end of every applicant  $a$ 's preference list. Let  $s_G(a)$  denote the set of top-ranked posts in  $a$ 's preference list that are even in  $G_1$ . We say that  $p$  is an  $s$ -post if  $p \in s_G(a)$  for some applicant  $a \in A$ . By appending last resort posts, every applicant has at least one choice that nobody else competes with him/her, thus  $s_G(a) \neq \emptyset$ .

The *reduced graph*  $G' = (A \cup P, E')$  is a subgraph of  $G$ , where  $E' = \{(a, p) : p \in f_G(a) \text{ or } p \in s_G(a)\}$ , that is,  $G'$  contains only those edges connecting  $a$  to its  $f$ -posts and  $s$ -posts for each applicant  $a$ . The *neighborhood*  $N_G(a)$  of a vertex  $a$  in  $G$  is the set of vertices adjacent to  $a$ . For simplicity, we denote  $\bigcup_{x \in X} N_G(x)$  as  $N_G(X)$ . For any matching  $M$  of  $G'$ ,  $M$  is said to be *applicant-complete* if all applicants are matched in  $M$ . Abraham *et al.* showed in [1, Theorem 3.1] a necessary and sufficient condition for a matching to be popular of  $G$ . Moreover, in the process of computing a popular matching of  $G$ , they further modified  $G'$  by removing all edges of types  $\mathcal{O}_{G_1}$ - $\mathcal{O}_{G_1}$ ,  $\mathcal{O}_{G_1}$ - $\mathcal{U}_{G_1}$ , and  $\mathcal{U}_{G_1}$ - $\mathcal{O}_{G_1}$ . We call such a subgraph the *simplified reduced graph* of  $G$  and denote it by  $G''$ . To pave a way for the proofs later in Section 4, we extend the result of [1, Theorem 3.1] as in Lemma 1. The proof is omitted here.

**Lemma 1.** *The following statements are equivalent.*

- a.  $M$  is a popular matching of  $G$ .
- b.  $M \cap E_1$  is a maximum matching of  $G_1$ , and  $M$  is applicant-complete in  $G'$ .
- c.  $M \cap E_1$  is a maximum matching of  $G_1$ , and  $M$  is applicant-complete in  $G''$ .

### 3 Inapproximability Results

In this section, we show the complexity results of the generalized popular condensation problem. Some ideas are inspired from [13]. We proceed with the decision version of this problem, formally stated as follows: Given an instance graph  $G = (A \cup P, E)$  and a nonnegative value  $\kappa$ , determine whether  $G$  has a condensing set with cost at most  $\kappa$ . This decision problem is abbreviated as GPC.

#### 3.1 The NP-Completeness

We start by arguing that the GPC problem belongs to NP. Given an instance graph and a value  $\kappa$ , a certificate consisting of a set of applicants, a set of posts, and a set of preference lists can be verified by checking if it belongs to  $\text{Cond}(G)$ , and checking whether the cost of the corresponding condensing set is at most  $\kappa$ . This process can certainly be done in polynomial time.

To prove the NP-hardness, we show a reduction from the monotone one-in-three SAT problem, which is a variant of SAT such that each variable occurs only unnegated in all clauses and each clause contains exact three literals. The monotone one-in-three SAT problem asks if there is a satisfying assignment to the variables such that only one literal is set to true in each clause. The monotone one-in-three SAT problem is proved to be NP-hard by Schaefer [21].

For convenience, we define some notations here only for this section. Let  $\phi$  be an instance of the monotone one-in-three SAT problem, where  $C_1, C_2, \dots, C_\mu$  are the clauses and  $X_1, X_2, \dots, X_\nu$  are the variables in  $\phi$ . Let  $c_j$  denote the number of clauses in which  $X_j$  appears. We construct an instance graph  $G$  of GPC from  $\phi$  as follows.

For each variable  $X_j$ , we have  $c_j$  global applicants  $A_j = \{x_j^{(k)} : k = 1, 2, \dots, c_j\}$  and  $c_j$  global posts  $P_j = \{u_j^{(k)} : k = 1, 2, \dots, c_j\}$ . The preference list of each global applicant  $x_j^{(k)}$  is of length one, and consists of the post  $u_j^{(k)}$ , where  $1 \leq k \leq c_j$  (see Figure 1). The cost of each global applicant is set to be one. Let clause  $C_i$  be  $(X_{j_1}, X_{j_2}, X_{j_3})$ , where  $j_1, j_2, j_3 \in \{1, 2, \dots, \nu\}$ . For this clause, six local applicants  $B_i = \{a_i^{(1)}, a_i^{(2)}, a_i^{(3)}, a_i^{(4)}, a_i^{(5)}, a_i^{(6)}\}$  and three local posts  $Q_i = \{p_i, q_i, r_i\}$  are introduced. Posts  $p_i$  and  $q_i$  belong to the preference lists of applicants in  $\{a_i^{(1)}, a_i^{(2)}, a_i^{(3)}\}$ , whereas  $r_i$  belongs to the preference lists of applicants in  $\{a_i^{(4)}, a_i^{(5)}, a_i^{(6)}\}$ . Additionally, global posts  $P_{j_1}, P_{j_2}$ , and  $P_{j_3}$  also belong to some

$x_j^{(1)}$	$u_j^{(1)}$
$x_j^{(2)}$	$u_j^{(2)}$
$\vdots$	$\vdots$
$x_j^{(c_j)}$	$u_j^{(c_j)}$

**Fig. 1.** Preference lists of the  $c_j$  global applicants in  $A_j$ . The cost of each global applicant is one.

$a_i^{(1)}$	$p_i \prec u_{j_1}^{(1)} \simeq u_{j_1}^{(2)} \simeq \dots \simeq u_{j_1}^{(c_{j_1})} \prec q_i$	$a_i^{(4)}$	$r_i \prec u_{j_1}^{(1)} \simeq u_{j_1}^{(2)} \simeq \dots \simeq u_{j_1}^{(c_{j_1})}$
$a_i^{(2)}$	$p_i \prec u_{j_2}^{(1)} \simeq u_{j_2}^{(2)} \simeq \dots \simeq u_{j_2}^{(c_{j_2})} \prec q_i$	$a_i^{(5)}$	$r_i \prec u_{j_2}^{(1)} \simeq u_{j_2}^{(2)} \simeq \dots \simeq u_{j_2}^{(c_{j_2})}$
$a_i^{(3)}$	$p_i \prec u_{j_3}^{(1)} \simeq u_{j_3}^{(2)} \simeq \dots \simeq u_{j_3}^{(c_{j_3})} \prec q_i$	$a_i^{(6)}$	$r_i \prec u_{j_3}^{(1)} \simeq u_{j_3}^{(2)} \simeq \dots \simeq u_{j_3}^{(c_{j_3})}$

**Fig. 2.** Preference lists of the six local applicants in  $B_i$ . The cost of each local applicant is  $\mu + 1$ .

preference lists of applicants in  $B_i$ . The preference lists of applicants in  $B_i$  are shown in Figure 2. The cost of each local applicant is set to be  $\mu + 1$ .

To sum up, according to the reduction above, we have an instance graph  $G = (A \cup P, E)$ , where the set of applicants  $A$  is the union of  $\bigcup_{j \in \{1, 2, \dots, \nu\}} A_j$  and  $\bigcup_{i \in \{1, 2, \dots, \mu\}} B_i$ , and the set of posts  $P$  is the union of  $\bigcup_{j \in \{1, 2, \dots, \nu\}} P_j$  and  $\bigcup_{i \in \{1, 2, \dots, \mu\}} Q_i$ .

*Property 1.*  $G$  does not admit a popular matching.

Since each global post  $u_j^{(k)}$  is the unique top-ranked post for exact one global applicant  $x_j^{(k)}$ , where  $1 \leq k \leq c_j$ , the global posts are unreachable in the first-choice graph  $G_1$  of  $G$ . Consider the local applicants in  $B_i$ : for each applicant  $a \in \{a_i^{(1)}, a_i^{(2)}, a_i^{(3)}\}$ , we have  $f_G(a) = \{p_i\}$  and  $s_G(a) = \{q_i\}$ . It shows that  $a_i^{(1)}$ ,  $a_i^{(2)}$ , and  $a_i^{(3)}$  compete for only two posts in  $G'$ . Since a popular matching must be an applicant-complete matching of  $G'$  (refer to Lemma 1),  $G$  does not admit a popular matching. As stated in Lemma 2,  $\phi$  is one-in-three satisfiable if and only if  $G$  has a popular condensation with cost at most  $\mu$ .

**Lemma 2.** *There is a one-in-three satisfying assignment for  $\phi$  if and only if  $G$  has a condensing set with cost at most  $\mu$ .*

Here we conclude this section with the following theorem.

**Theorem 1.** *The generalized popular condensation problem is NP-hard.*

### 3.2 The Inapproximability

We show that the generalized popular condensation problem is NP-hard to approximate to within a factor of  $\alpha^* - \epsilon$ , where  $\alpha^* = \max\{n/11, m/2, \sqrt{W}/2\}$ ,

$x_j^{(1)}$	$u_j^{(1)}$
$x_j^{(2)}$	$u_j^{(2)}$
$\vdots$	$\vdots$
$x_j^{((\mu^3+1)c_j)}$	$u_j^{((\mu^3+1)c_j)}$

**Fig. 3.** Preference lists of the  $(\mu^3 + 1)c_j$  global applicants in  $A_j$ . The cost of each global applicant is one.

$a_i^{(1)} \mid p_i \prec u_{j_1}^{(1)} \simeq \dots \simeq u_{j_1}^{((\mu^3+1)c_{j_1})} \prec q_i$	$a_i^{(3t+1)} \mid r_i^t \prec u_{j_1}^{(1)} \simeq \dots \simeq u_{j_1}^{((\mu^3+1)c_{j_1})}$
$a_i^{(2)} \mid p_i \prec u_{j_2}^{(1)} \simeq \dots \simeq u_{j_2}^{((\mu^3+1)c_{j_2})} \prec q_i$	$a_i^{(3t+2)} \mid r_i^t \prec u_{j_2}^{(1)} \simeq \dots \simeq u_{j_2}^{((\mu^3+1)c_{j_2})}$
$a_i^{(3)} \mid p_i \prec u_{j_3}^{(1)} \simeq \dots \simeq u_{j_3}^{((\mu^3+1)c_{j_3})} \prec q_i$	$a_i^{(3t+3)} \mid r_i^t \prec u_{j_3}^{(1)} \simeq \dots \simeq u_{j_3}^{((\mu^3+1)c_{j_3})}$

**Fig. 4.** Preference lists of the  $3\mu^3 + 6$  local applicants in  $B_i$ ,  $t = 1, 2, \dots, \mu^3 + 1$ . The cost of each local applicant is  $\mu^3 + 1$ .

$\epsilon > 0$  is any constant,  $n$  is the number of applicants and posts,  $m$  is the total length of preference lists, and  $W$  is the sum of costs of all applicants. Again, we give a reduction from the monotone one-in-three SAT problem here. Given an instance  $\phi$  of the monotone one-in-three SAT problem, we construct an instance  $G = (A \cup P, E)$  of GPC in a manner similar to that in Section 3.1. Only the differences are pointed out as follows.

- For each variable  $X_j$ , instead of having  $c_j$  global applicants and posts as in Section 3.1, now we have  $(\mu^3 + 1)c_j$  global applicants in  $A_j = \{x_j^{(k)} : k = 1, 2, \dots, (\mu^3 + 1)c_j\}$  and equal-sized global posts in  $P_j = \{u_j^{(k)} : k = 1, 2, \dots, (\mu^3 + 1)c_j\}$ . The preference list of each global applicant  $x_j^{(k)}$  is of length one, and consists of the post  $u_j^{(k)}$ , where  $1 \leq k \leq (\mu^3 + 1)c_j$ .
- For each clause  $C_i = (X_{j_1}, X_{j_2}, X_{j_3})$ , instead of having six local applicants and three local posts, here we have  $3\mu^3 + 6$  local applicants in  $B_i = \{a_i^{(k)} : k = 1, 2, \dots, 3\mu^3 + 6\}$  and  $\mu^3 + 3$  local posts in  $Q_i = \{p_i, q_i\} \cup \{r_i^{(k)} : k = 1, 2, \dots, \mu^3 + 1\}$ .
- Regarding the costs of applicants, global applicants are still appended with cost one, whereas local applicants are appended with cost  $\mu^3 + 1$ .

The preference lists of applicants in  $A_j$  and  $B_i$  are illustrated in Figure 3 and Figure 4, respectively.

**Lemma 3.** *If there is a one-in-three satisfying assignment for  $\phi$ ,  $G$  has a condensing set of cost  $\mu$ . Otherwise, the cost of any condensing set of  $G$  is at least  $\mu^3 + 1$ .*



Due to the space limitation, we omit the proof of Lemma 3 here. Based on the reduction, for  $\mu > 3$ , we have

- (1) the number of applicants and posts in  $G$  is  $n = 2(\mu^3 + 1) \sum_{j=1,2,\dots,\nu} c_j + \mu(4\mu^3 + 9) = 2(\mu^3 + 1)(3\mu) + \mu(4\mu^3 + 9) = 10\mu^4 + 15\mu \leq 11\mu^4$ ;
- (2) the total length of the preference lists is  $m = (\mu^3 + 1) \sum_{j=1,2,\dots,\nu} c_j + \mu(6 + (\mu^3 + 1)) + \sum_{j=1,2,\dots,\nu} c_j^2(\mu^3 + 1) \leq (\mu^3 + 1)(3\mu) + (\mu^4 + 7\mu) + (\mu^6 + \mu^3) = \mu^6 + 4\mu^4 + \mu^3 + 10\mu \leq 2\mu^6$ ; and
- (3) the sum of costs of all applicants is  $W = (\mu^3 + 1) \sum_{j=1,2,\dots,\nu} c_j + \mu(3\mu^3 + 6)(\mu^3 + 1) = 3\mu^7 + 12\mu^4 + 10\mu \leq 4\mu^7$ .

**Theorem 2.** *The generalized popular condensation problem is NP-hard to approximate to within a factor of  $\alpha$ , where  $\alpha = \max\{(n/11)^{\frac{1}{2}}, (m/2)^{\frac{1}{3}}, (W/4)^{\frac{2}{7}}\}$ ,  $n$  is the number of applicants and posts,  $m$  is the total length of preference lists, and  $W$  is the sum of costs of all applicants.*

*Proof.* We only show that there is no polynomial-time  $(n/11)^{\frac{1}{2}}$ -approximation algorithm for the generalized popular condensation problem, unless  $\text{NP} = \text{P}$ . For  $(m/2)^{\frac{1}{3}}$  and  $(W/4)^{\frac{2}{7}}$ , the results can be derived in a similar way.

Suppose to the contrary that there is a  $(n/11)^{\frac{1}{2}}$ -approximation algorithm  $\mathbf{X}$ . If  $\phi$  is one-in-three satisfiable, by Lemma 3,  $\mathbf{X}$  returns a popular condensation of  $G$  with cost at most  $(\sqrt{n/11})\mu = (\sqrt{11\mu^4/11})\mu = \mu^3$ . If  $\phi$  is not one-in-three satisfiable, all popular condensations of  $G$  are of cost greater than  $\mu^3$ . Hence,  $\mathbf{X}$  returns an answer of cost at least  $\mu^3 + 1$ . As a result, we can use  $\mathbf{X}$  and the reduction mentioned in this subsection to determine whether  $\phi$  is one-in-three satisfiable in polynomial time. □

In fact, for any given  $\epsilon > 0$ , by simply increasing the number of global applicants in  $A_j$  to  $(\mu^\beta + 1)c_j$ , and the number of local applicants in  $B_i$  to  $3\mu^\beta + 6$ , with  $\beta = \lceil \frac{2-\epsilon}{\epsilon} \rceil$ , we can derive that the problem cannot be approximated to within a factor of  $(n/11)^{1-\epsilon}$  in polynomial time, unless  $\text{NP} = \text{P}$ . Similarly, by setting  $\beta$  to be  $\lceil \frac{4-3\epsilon}{\epsilon} \rceil$  and  $\lceil \frac{3-\epsilon}{2\epsilon} \rceil$ , the factor can be refined as  $(m/2)^{1-\epsilon}$  and  $(\sqrt{W}/2)^{1-\epsilon}$ , respectively. We summarize the results in Corollary 1. Notice that a similar analysis can be applied to [13] to improve its inapproximability result.

**Corollary 1.** *The generalized popular condensation problem is NP-hard to approximate to within a factor of  $\alpha^* - \epsilon$ , where  $\alpha^* = \max\{n/11, m/2, \sqrt{W}/2\}$ ,  $\epsilon > 0$  is any constant,  $n$  is the number of applicants and posts,  $m$  is the total length of preference lists, and  $W$  is the sum of costs of all applicants.*

## 4 A special Case Where Applicants' Costs Are Equal

In this section, we consider the case where applicants' costs are equal. We show that the optimal solution can be computed in  $O(\sqrt{nm})$  time by modifying the

algorithm given in [1]. The modified algorithm is given in Section 4.2, and the correctness is based on the lower bound on  $|D^*|$  given in Section 4.1, where  $D^*$  is the condensing set w.r.t. an optimal popular condensation.

### 4.1 A Lower Bound on $|D^*|$

Given an instance in which applicants' costs are equal, let  $H^*$  be an optimal solution of the generalized popular condensation problem. Let the elements in  $A \setminus A_{H^*}$  be ordered arbitrarily as  $a_1, a_2, \dots, a_k$ , where  $k = |A \setminus A_{H^*}|$ , and let  $H^{(i)}$  be an instance graph  $(A^{(i)} \cup P, E)$ , where  $A^{(i)} = A \setminus \{a_1, \dots, a_i\}$ . Obviously, by definition, no matter how the elements in  $A \setminus A_{H^*}$  are ordered, the instance graph  $H^{(i)}$  admits no popular matching unless  $i = k$ .

Let  $M''_G$  be a maximum matching of  $G''$ . In this subsection, we claim that  $|A| - |M''_G|$  is a lower bound on  $k$ . According to Lemma 1, there is an applicant-complete matching in  $H^*$ , and such a matching is a maximum matching of  $H^*$ . If  $|A| - |M''_G| > k$ , then by the pigeonhole principle, there must be some  $i$  satisfying  $|M''_{H^{(i-1)}}| < |M''_{H^{(i)}}|$ . Assume that  $G = H^{(0)}$ , our strategy to prove the claim is to show  $|M''_{H^{(i-1)}}| \geq |M''_{H^{(i)}}|$ , for  $1 \leq i \leq k$ .

Let  $G$  be a bipartite graph, and let  $H = G - a$  be the subgraph of  $G$  in which vertex  $a$  is removed. In dealing with the Gallai-Edmonds decomposition of  $G$  and  $H$ , Lovász and Plummer [14] gave a useful lemma that describes how the Gallai-Edmonds decomposition changes from  $G$  to  $H$ . The lemma is called the *stability lemma*, as shown in Lemma 4.

**Lemma 4 (the Stability Lemma [14]).** *Let  $G$  be a bipartite graph,  $\{\mathcal{E}_G, \mathcal{O}_G, \mathcal{U}_G\}$  be the Gallai-Edmonds decomposition, and  $u$  be a vertex of  $G$ .*

- a. *If  $u \in \mathcal{O}_G$ , then  $\mathcal{O}_{G-u} = \mathcal{O}_G \setminus \{u\}$ ,  $\mathcal{U}_{G-u} = \mathcal{U}_G$  and  $\mathcal{E}_{G-u} = \mathcal{E}_G$ .*
- b. *If  $u \in \mathcal{U}_G$ , then  $\mathcal{O}_{G-u} \supseteq \mathcal{O}_G$ ,  $\mathcal{U}_{G-u} \subseteq \mathcal{U}_G \setminus \{u\}$  and  $\mathcal{E}_{G-u} \supseteq \mathcal{E}_G$ .*
- c. *If  $u \in \mathcal{E}_G$ , then  $\mathcal{O}_{G-u} \subseteq \mathcal{O}_G$ ,  $\mathcal{U}_{G-u} \supseteq \mathcal{U}_G$ , and  $\mathcal{E}_{G-u} \subseteq \mathcal{E}_G \setminus \{u\}$ .*

Based on the stability lemma, we may further characterize the Gallai-Edmonds decompositions of  $G$  and  $H$  by Lemmata 5 and 6. Due to the space limitation, we omit the proofs of the following lemmata.

**Lemma 5.** *Let  $G = (A \cup P, E)$  be a bipartite graph, and let  $H = G - a$  for some  $a \in A$ . If  $a \in \mathcal{E}_G$ , then  $\mathcal{E}_H \cap P = \mathcal{E}_G \cap P$  and  $\mathcal{O}_H \cap A = \mathcal{O}_G \cap A$ .*

**Lemma 6.** *Let  $a \in \mathcal{U}_{G_1}$ . If  $v \in \mathcal{O}_{H_1} \setminus \mathcal{O}_{G_1}$ , then  $N_{H''}(v) \subseteq \mathcal{E}_{H_1} \setminus \mathcal{E}_{G_1}$ . Moreover, there is a matching in  $G''$  that matches all vertices in  $\{a\} \cup (\mathcal{O}_{H_1} \setminus \mathcal{O}_{G_1})$  to  $\mathcal{E}_{H_1} \setminus \mathcal{E}_{G_1}$ .*

To prove the claim given at the beginning of this subsection, that is,  $|A| - |M''_G|$  is a lower bound on  $|D^*|$ , it suffices to prove Lemma 7.

**Lemma 7.** *Let  $H = G - a$  where  $a \in A$ , and let  $M''_H$  and  $M''_G$  be maximum matchings of  $H''$  and  $G''$ , respectively. We have that  $|M''_H| \leq |M''_G|$ .*

### 4.2 Computing an Optimal Condensing Set

Given any instance graph, an optimal condensing set can be computed by modifying the algorithm proposed by Abraham *et al.* [1] as follows.

---

**Algorithm 1.** POPULARCONDENSATION( $G = (A \cup P, E)$ )

---

- 1 Compute  $G_1 = (A \cup P, E_1)$ , where  $E_1 = \{(a, p) : r_G(a, p) = 1\}$ ;
  - 2 Compute a maximum matching  $M_1$  of  $G_1$ ;
  - 3 Compute the simplified reduced graph  $G''$ ;
  - 4 Compute a maximum matching  $M$  of  $G''$  by augmenting  $M_1$ ;
  - 5  $D :=$  the set of applicants in  $A$  not matched by  $M$ ;
  - 6 **return**  $D$ ;
- 

To show the correctness, we claim that (i)  $M$  is a popular matching of  $H = ((A \setminus D) \cup P, E_H)$ , where  $E_H = \{(a, p) \in E : a \in A \setminus D\}$ ; (ii)  $D$  is a condensing set of minimum cardinality. Condition (ii) is shown in Section 4.1. To show that Condition (i) holds, we proceed with the following lemma.

**Lemma 8.** *For an instance graph  $G = (A \cup P, E)$  in which applicants’ costs are equal, POPULARCONDENSATION( $G$ ) computes a condensing set w.r.t. an optimal popular condensation of  $G$ .*

*Proof.* Let  $D$  be the output of POPULARCONDENSATION( $G$ ),  $M$  be the corresponding matching, and  $H = (A_H \cup P, E_H)$  be the instance graph with  $A_H = A \setminus D$ ,  $E_H = E \setminus \{(a, p) : a \in D\}$ , and  $r_H(e) = r_G(e)$ . Let  $H_1 = (A_H \cup P, E_{H_1})$  be the first-choice graph of  $H$  and  $H'$  be the reduced graph of  $H$ . By Lemma 1, it suffices to show that  $M \cap E_{H_1}$  is a maximum matching of  $H_1$ , and  $M$  is an applicant-complete matching of  $H'$ .

First, we show that  $M \cap E_{H_1}$  is a maximum matching of  $H_1$ . Let  $|M|$  denote the size of the matching  $M$ . Suppose that  $M_{H_1}$  and  $M_{G_1}$  are maximum matchings of  $H_1$  and  $G_1$ , respectively. Since  $H_1$  is a subgraph of  $G_1$ , we have  $|M_{H_1}| \leq |M_{G_1}|$ . As shown in [1, Lemma 3.6],  $M \cap E_1$  is a maximum matching of  $G_1$ , and thus we have

$$|M \cap E_1| = |M \cap E_{H_1}| \leq |M_{H_1}| \leq |M_{G_1}| = |M \cap E_1|.$$

Therefore, the equality  $|M \cap E_{H_1}| = |M_{H_1}|$  holds, and  $M \cap E_{H_1}$  is a maximum matching of  $H_1$ .

Next, we show that  $M$  is an applicant-complete matching of  $H'$ . For each  $(a, p) \in M$ , if  $p$  is an  $f$ -post of  $a$ , then  $(a, p)$  is still an edge of  $H'$ . Otherwise,  $p$  is an  $s$ -post of  $a$  in  $G$ . Since  $M$  is obtained by augmenting an maximum matching of  $G_1$ , all vertices in  $\mathcal{U}_{G_1}$  are matched in  $M$ , and thus  $D$  consists of only applicants in  $\mathcal{O}_{G_1} \cup \mathcal{E}_{G_1}$ . By the stability lemma and Lemma 5,  $\mathcal{E}_{H_1} \cap P = \mathcal{E}_{G_1} \cap P$ , and therefore  $(a, p) \in E(H')$ . As a result,  $M$  is a subgraph of  $H'$ , and we can conclude that  $M$  is applicant-complete in  $H'$ . □

For the time complexity of  $\text{POPULARCONDENSATION}(G)$ , the most time-consuming steps are to compute the maximum matchings of  $G_1$  and  $G''$ , which can be done in  $O(\sqrt{nm})$  by the algorithm of Hopcroft and Karp [8]. Together with the lower bound analyzed in Section 4.1, we conclude this section by the following theorem.

**Theorem 3.** *Given that all applicants' costs are equal, the generalized popular condensation problem can be solved in  $O(\sqrt{nm})$  time.*

## 5 Concluding Remarks

This paper generalizes the result of the original popular condensation problem [22]. We show that when applicants are appended with costs and the preference lists may contain ties, it is NP-hard to compute a condensing set with minimum cost such that the resulting instance admits a popular matching. We further show that it is in fact NP-hard to approximate the problem within a factor of  $\alpha^* - \epsilon$ , where  $\alpha^* = \max\{n/11, m/2, \sqrt{W}/2\}$ , and  $\epsilon > 0$  is any constant. For the special case where all applicants' costs are equal, we show that computing an optimal condensing set can be done in  $O(\sqrt{nm})$  time. Note that since the generalized popular condensation problem under the special case is at least as hard as the popular matching problem, which is equivalent to the maximum-cardinality bipartite matching problem [1], all these problems have equivalent time complexity.

**Acknowledgments.** The authors would like to thank the anonymous reviewers for helpful comments. Yen-Wei Wu, Wei-Yin Lin, and Kun-Mao Chao were supported in part by MOST grants 101-2221-E-002-063-MY3 and 103-2221-E-002-157-MY3, and Hung-Lung Wang was supported in part by MOST grant 103-2221-E-141-004 from the Ministry of Science and Technology, Taiwan.

## References

1. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. *SIAM Journal on Computing* **37**(4), 1030–1045 (2007)
2. Abraham, D.J., Kavitha, T.: Dynamic matching markets and voting paths. In: *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory*, pp. 65–76 (2006)
3. Biró, P., Irving, R.W., Manlove, D.F.: Popular matchings in the marriage and roommates problems. In: Calamoneri, T., Diaz, J. (eds.) *CIAC 2010*. LNCS, vol. 6078, pp. 97–108. Springer, Heidelberg (2010)
4. Gardenfors, P.: Match Making: assignments based on bilateral preferences. *Behavioural Sciences* **20**, 166–173 (1975)
5. Pulleyblank, W.R.: Matchings and Extensions. In: Graham, R.L., Grötschel, M., Lovasz, L. (eds.): *The Handbook of Combinatorics*, ch. 3, pp. 179–232. MIT Press, Cambridge (1995)
6. Huang, C.-C., Kavitha, T.: Near-popular matchings in the roommates problem. *SIAM Journal on Discrete Mathematics* **27**(1), 43–62 (2013)

7. Huang, C.-C., Kavitha, T.: Popular matchings in the stable marriage problem. *Information and Computation* **222**, 180–194 (2013)
8. Hopcroft, J.E., Karp, R.M.: A  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing* **2**(4), 225–231 (1973)
9. Huang, C.-C., Kavitha, T., Michail, D., Nasre, M.: Bounded unpopularity matching. In: *Proceedings of the 12th Scandinavian Workshop on Algorithm Theory*, pp. 127–137 (2008)
10. Kavitha, T.: Popularity vs maximum cardinality in the stable marriage setting. In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 123–134 (2012)
11. Kavitha, T., Nasre, M.: Optimal popular matchings. *Discrete Applied Mathematics* **157**(14), 3181–3186 (2009)
12. Kavitha, T., Nasre, M.: Popular matchings with variable item copies. *Theoretical Computer Science* **412**, 1263–1274 (2011)
13. Kavitha, T., Nasre, M., Nimbhorkar, P.: Popularity at minimum cost. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) *ISAAC 2010, Part I. LNCS*, vol. 6506, pp. 145–156. Springer, Heidelberg (2010)
14. Lovasz, L., Plummer, M.D.: *Matching Theory*. North-Holland (1986)
15. Mahdian, M.: Random popular matchings. In: *Proceedings of the 7th ACM Conference on Electronic Commerce*, pp. 238–242 (2006)
16. Manlove, D.F., Sng, C.T.S.: Popular matchings in the capacitated house allocation problem. In: *Proceedings of the 14th Annual European Symposium on Algorithms*, pp. 492–503 (2006)
17. McCutchen, R.M.: The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) *LATIN 2008. LNCS*, vol. 4957, pp. 593–604. Springer, Heidelberg (2008)
18. McDermid, E., Irving, R.W.: Popular matchings: structure and algorithms. *Journal of Combinatorial Optimization* **22**(3), 339–358 (2011)
19. Mestre, J.: Weighted popular matchings. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) *ICALP 2006. LNCS*, vol. 4051, pp. 715–726. Springer, Heidelberg (2006)
20. Paluch, K.: Popular and clan-popular  $b$ -matchings. In: *Proceedings of the 23rd International Symposium on Algorithms and Computation*, pp. 116–125 (2012)
21. Schaefer, T.J.: The complexity of satisfiability problems. In: *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, New York, pp. 216–226 (1978)
22. Wu, Y.-W., Lin, W.-Y., Wang, H.-L., Chao, K.-M.: An optimal algorithm for the popular condensation problem. In: Lecroq, T., Mouchard, L. (eds.) *IWOCA 2013. LNCS*, vol. 8288, pp. 412–422. Springer, Heidelberg (2013)

# **Graph Algorithms: Approximation II**

# Dirichlet Eigenvalues, Local Random Walks, and Analyzing Clusters in Graphs

Pavel Kolev<sup>(✉)</sup> and He Sun

Max Planck Institute für Informatik, 66123 Saarbrücken, Germany  
{pkolev,hsun}@mpi-inf.mpg.de

**Abstract.** A cluster  $S$  in a massive graph  $G$  is characterised by the property that its corresponding vertices are better connected with each other, in comparison with the other vertices of the graph. Modeling, finding and analyzing clusters in massive graphs is an important topic in various disciplines. In this work we study local random walks that always stay in a cluster  $S$ . Moreover, we initiate the study of the local mixing time and the almost stable distribution, by analyzing Dirichlet eigenvalues in graphs. We prove that the Dirichlet eigenvalues of any connected subset  $S$  can be used to bound the  $\varepsilon$ -uniform mixing time, which improves the previous best-known result. We further present two applications of our results. The first is a polynomial-time algorithm for finding clusters with an improved approximation guarantee, while the second is the significance ordering of vertices in a cluster.

## 1 Introduction

Random walks are one of the fundamental tools in various disciplines such as mathematics, physics and computer science, and have been studied from different aspects (e.g., mixing time, cover time, relations to the eigenvalue gap of the stochastic matrix, conductance of graphs), and under many variations (e.g., multiple random walks [8], coalescing random walks [7], and deterministic random walks [9]). In computer science, random walks are widely used in designing distributed, randomized and local algorithms, in which the global information of the graph is unknown to every vertex, and vertices can only exchange information with their neighbors. Some specific problems in which random walks are used include graph exploration [3], load balancing [15], and local algorithms for finding clusters [10].

The most widely studied form of random walks is the *lazy random walk*. Let  $G = (V, E)$  be an undirected and connected graph, and  $\deg(u)$  be the degree of vertex  $u$ . A lazy random walk in  $G$  is a sequence of random vertices  $X_0, X_1, \dots, X_t$  modelled according to the following rule. There is an initial vertex  $X_0 \in V$  from which the random walk starts, and in each step  $i \geq 1$  every vertex  $X_{i-1} = u$

---

This work has been partially funded by the Cluster of Excellence “Multimodal Computing and Interaction” within the Excellence Initiative of the German Federal Government.

stays at the current vertex  $u$  with probability  $1/2$  or moves to a random neighbor with probability  $1/2\text{deg}(u)$ .

In the this paper we study a *local random walk*. Let  $S \subseteq V$  be a subset with  $\partial S \neq \emptyset$ , where  $\partial S \triangleq \{(u, v) \mid u \in S \text{ and } v \in V \setminus S\}$  and consider a random process  $X_0, \dots, X_t$  in which every  $X_i \in S$  for  $1 \leq i \leq t$ , i.e., the lazy random walk always stays inside  $S$  during the first  $t$  steps. Of particular interest is the case when vertices in  $S$  are well connected, in contrast to the vertices between  $S$  and  $V \setminus S$ . In this case, the set  $S$  corresponds to a cluster in social networks, where the connections between people inside a cluster are substantially more, in comparison with the connections between different clusters. Theoretical studies on local random walks have significantly influenced the design of local algorithms for finding well-connected clusters [1, 2, 10, 16, 17].

We first address several theoretical aspects on local random walks. One basic and well-known result on lazy random walks is that after a certain number of steps, the distribution of  $X_t$  in step  $t$  becomes *stable*, i.e. the probability that vertex  $u$  is visited is approximately proportional to  $\text{deg}(u)$ . In contrast, the situation for local random walks is much more complicated. Since  $\partial S \neq \emptyset$ , in each step the random walk leaves  $S$  with (probably) non-zero probability, and this probability changes with respect to the current location  $u$  of the random walk. The probability that the random walk leaves  $S$  in the next step is proportional to the number of neighbors of  $u$  outside  $S$  and  $\text{deg}(u)$ . Interestingly, we show that the phenomenon of *stability* exists even for local random walks. Our first result is summarized in Theorem 1, and we refer the reader to Theorem 6 for a precise and formal statement.

**Theorem 1 (Informal).** *Let  $G$  be an undirected and connected graph, and  $S$  a subset of vertices. Let  $\mathcal{P}_{S,w}[X_t = v]$  be the probability that a lazy random walk starting from  $w \in S$  stays in  $S$  during the first  $t$  steps and reaches  $v$  in step  $t$ . Then, for any  $\varepsilon$  there is a step  $\text{MIX}_S(\varepsilon)$ , called the local mixing time, such that it holds for any  $u, v \in S$  and  $t \geq \text{MIX}_S(\varepsilon)$  that*

$$(1 - \varepsilon') \cdot \frac{(\mathbf{p}_S)_u}{(\mathbf{p}_S)_v} \leq \frac{\mathcal{P}_{S,w}[X_t = u]}{\mathcal{P}_{S,w}[X_t = v]} \leq (1 + \varepsilon') \cdot \frac{(\mathbf{p}_S)_u}{(\mathbf{p}_S)_v},$$

where  $\varepsilon' = 2\varepsilon/(1-\varepsilon)$  and  $\mathbf{p}_S$  is the almost stable distribution that is independent of step  $t$ .

Since the random walk may leave  $S$  with non-zero probability in each step and there is no “real” stationary distribution for local random walks, in Theorem 1 we study the ratio between the probabilities of the local random walk visiting different vertices  $u, v \in S$  that start from the same initial vertex  $w \in S$ . We prove that this ratio is preserved after  $\text{MIX}_S(\varepsilon)$  steps. Equivalently, once  $\text{MIX}_S(\varepsilon)$  steps are performed the “shape” of the resulting distribution becomes stable, and this *almost stable distribution* can be precisely characterized by Dirichlet eigenvalues and the associated eigenvectors. When  $S = V$ , the almost stable distribution coincides with the standard stationary distribution, and  $\text{MIX}_S(\varepsilon)$  matches the



well-known bound on the uniform mixing time, implying that our result is a natural generalization of the result for global random walks.

We further address the influence of the structure of  $S$  on the mixing rate of (non-local) lazy random walks. Formally, we study the  $\varepsilon$ -uniform mixing time  $\tau(\varepsilon)$  [11], which is defined by

$$\tau(\varepsilon) \triangleq \min \left\{ t : \left| \frac{\mathcal{P}_u[X_t = v] - \pi_v}{\pi_v} \right| \leq \varepsilon, \forall u, v \in V \right\}. \tag{1}$$

Here  $\pi \in \mathbb{R}^V$  is the stationary distribution of a lazy random walk in  $G$ , and  $\mathcal{P}_u[X_t = v]$  is the probability that a  $t$ -step random walk starting from  $u$  reaches  $v$ . We prove that for any connected subset  $S \subset V$ , the Dirichlet eigenvalue  $\lambda_S$  (i.e., the largest eigenvalue of matrix  $M_S^1$ ) gives a lower bound on  $\tau(\varepsilon)$ . This bound presents an improvement over the previous best result [10].

**Theorem 2.** *Let  $G = (V, E)$  be an undirected and connected graph. For  $1 \leq \gamma \leq \text{vol}(V)/2$ ,  $0 < \varepsilon < 1$ , and any subset  $S \subset V$  satisfying  $\text{vol}(S) \leq \gamma$ ,  $G[S]$  is connected and  $\lambda_S \geq 1/4$ , it holds that  $\tau(\varepsilon) \geq [2(1 - \lambda_S)]^{-1} \cdot \ln(\text{vol}(V)/2\gamma)$ .*

Our results have several practical implications. First, we present an improved algorithm for finding well-connected clusters. Our result is summarized in Theorem 3, and the algorithm is presented in Section 6.

**Theorem 3.** *There is a polynomial-time algorithm that takes as input an undirected and connected graph  $G = (V, E)$  with minimum degree at least 4, a target largest eigenvalue  $\lambda$  and a real number  $0 < \varepsilon < 1/2$ , and outputs a set  $S$  satisfying the following condition: If  $\lambda_{\max}(M_U) \geq \lambda$ , for some connected subset  $U \subset V$ , then  $\text{vol}(S) \leq 2 \text{vol}(U)^{1+\varepsilon}$  and  $\lambda_{\max}(M_S) \geq 1 - \sqrt{2(1 - \lambda)}/\varepsilon$ .*

Second, our result on almost stable distribution (Theorem 1) implies an ordering of vertices in  $S$  according to their significance. A vertex has high significance if a local random walk started from that vertex stays inside  $S$  with high probability. We prove that the vector representing the significance of vertices can be expressed as a scaled Dirichlet eigenvector. Our result is summarized as follows.

**Theorem 4 (Informal).** *Let  $G = (V, E)$  be an undirected and connected graph,  $S \subseteq V$  such that  $G[S]$  is connected, and  $\mathbf{v}$  is the eigenvector corresponding to the Dirichlet eigenvalue  $\lambda_S$  of matrix  $M_S$ . Then, the vector  $\mathbf{z}_S \triangleq D_S^{-1/2} \mathbf{v}$  gives significance ordering of vertices in  $S$ .*

Interestingly, the significance ordering of vertices in  $S$ , which is based on Dirichlet eigenvectors, corresponds to the so-called *sweep ordering* [1, 10, 13, 14] of the vector  $\mathbf{p}_S$ . In particular, it holds that  $(\mathbf{z}_S)_{u_1} \geq \dots \geq (\mathbf{z}_S)_{u_S}$  if and only if  $(\mathbf{p}_S)_{u_1} / \text{deg}(u_1) \geq \dots \geq (\mathbf{p}_S)_{u_1} / \text{deg}(u_S)$ . This suggests that further connections among finding significant vertices, Dirichlet eigenvectors and sweep sets may exist.

<sup>1</sup> Matrix  $M_S$  is a normalized adjacency sub-matrix consisting of rows and columns indexed by the vertices in  $S$ . We refer to Section 2 for all formal definitions.

*Related Work.* There are several studies that are closely related to our work. The first line involves spectral analysis of subgraphs. Chung and Yau [6] showed a relation between isoperimetric inequalities and Laplacian operators with Dirichlet conditions. Moreover, a relation between expansion of an induced subgraph and the Dirichlet eigenvalue, a.k.a. local Cheeger inequality, was shown in [5]. The second line deals with the design of local algorithms for finding well-connected clusters. Various local algorithms based on truncated random walks [12,16], PageRank [1,17], and Evolving Sets [2,10] have been proposed in the last decade.

## 2 Background Knowledge & Notations

Let  $G = (V, E)$  be an undirected and connected graph with  $n$  vertices. Let  $A$  be the adjacency matrix of graph  $G$ , and let  $D$  be the diagonal matrix consisting of degrees of vertices in  $G$ . The normalized adjacency matrix of graph  $G$  is defined by  $M \triangleq D^{-1/2}AD^{-1/2}$ . For any subset  $S \subseteq V$ , the induced subgraph determined by  $S$  has edge set consisting of all edges of  $G$  with both endpoints in  $S$ . We denote by  $G[S]$  the induced subgraph determined by  $S$ . The conductance of set  $S$  is defined by

$$\phi_S \triangleq \frac{|\partial S|}{\min\{\text{vol}(S), \text{vol}(V \setminus S)\}}, \tag{2}$$

where  $\text{vol}(S) \triangleq \sum_{v \in S} \text{deg}(v)$  is the volume of  $S$ . The conductance of a graph  $G$  is defined by

$$\phi_G \triangleq \min_{\substack{\emptyset \neq S \subset V \\ \text{vol}(S) \leq \text{vol}(G)/2}} \phi_S.$$

We further define the *local Cheeger constant* by

$$h_S \triangleq \min_{T \subset S} \frac{|\partial T|}{\text{vol}(T)},$$

and we call a set  $S$  with  $\text{vol}(S) \leq \text{vol}(V)/2$  a *cluster* if  $h_S = \phi_S$ , i.e.,  $S$  is the set such that all subsets  $T \subset S$  have better conductance than the set  $S$  itself.

We use  $M_S$  to represent the sub-matrix of  $M$  that consists of rows and columns indexed by the vertices in  $S$ . The matrices  $D_S$  and  $A_S$  are defined in the same way. The eigenvalues of an  $n$  by  $n$  matrix are expressed by  $\lambda_{\max}(\cdot) = \lambda_1(\cdot) \geq \lambda_2(\cdot) \geq \dots \geq \lambda_n(\cdot)$  in the non-increasing order.

We study the functions with the Dirichlet boundary conditions, i.e., we consider the space of functions  $\{f : S \cup \delta S \mapsto \mathbb{R}\}$  that satisfy the Dirichlet condition  $f(x) = 0$  for any  $x$  in the vertex boundary  $\delta S$  of  $S$ , where  $\delta S \triangleq \{u \notin S : (u, v) \in E \text{ and } v \in S\}$ . We use the notation  $f \in D^*$  to denote that  $f$  satisfies the Dirichlet boundary condition. By Courant-Fischer Formula, the largest eigenvalue  $\lambda_S$  of matrix  $M_S$ , can be written as

$$\lambda_S = \max_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \in D^*} \frac{\mathbf{x}^T M \mathbf{x}}{\mathbf{x}^T \mathbf{x}},$$

and we call the vector restricted to  $\mathbb{R}^S$  the Dirichlet eigenvector.

**Lemma 1** ([5]). *Let  $G$  be a graph, and  $S \subseteq V$  a subset such that  $G[S]$  is connected. Then, the Dirichlet eigenvalue satisfies*

$$1 - \phi_S \leq 1 - h_S \leq \lambda_S \leq 1 - h_S^2/2.$$

Notice that  $h_S = \phi_S$  and  $\lambda_S \leq 1 - \phi_S^2/2$ , if  $S$  is a cluster. For further discussion about the Dirichlet eigenvalues, we refer the reader to [4].

We use  $A, B, \dots$  to express matrices, and  $\mathbf{x}, \mathbf{y}, \dots$  for column vectors. We will write  $\mathbf{x} \in \mathbb{R}^V$  and  $A \in \mathbb{R}^{V \times V}$  to emphasise that they are indexed by the vertices of a graph  $G$ . We denote by  $\mathbf{1} \triangleq (1, 1, \dots, 1)^T$  the all one vector, whereas  $\mathbf{1}_S$  is the characteristic vector of a subset  $S \subset V$ . We abuse the notation and use  $\mathbf{1}_v$  instead of  $\mathbf{1}_{\{v\}}$ . We write  $\mathbf{p}(S) \triangleq \sum_{u \in S} \mathbf{p}_u$ , for a subset  $S \subseteq V$ .

### 3 The Staying Probability

We first consider the *lazy random walk* in  $G$  in which the random walk stays at the current vertex  $u$  with probability  $1/2$  or moves to a random neighbor with probability  $1/2\text{deg}(u)$ . This random walk is characterized by the transition matrix  $W \triangleq (I + D^{-1}A)/2$ . Let  $X_t$  be the vertex in which the random walk stays in step  $t$  and  $\pi$  be the stationary distribution of the random walk.

For any set  $S \subseteq V$ , vertex  $u$  and integer  $t \geq 0$  we write  $\text{rem}(u, t, S)$  to denote the probability that a lazy random walk starting from a vertex  $v$  always stays in  $S$  during the first  $t$  steps. The motivation behind studying  $\text{rem}(u, t, S)$  is to analyze the performance of a random walk when  $S$  is a cluster. In this case, we expect the random walk to stay entirely in  $S$  with reasonable probability for a relatively large number of steps  $t$ . Our result on staying probability is as follows.

**Theorem 5.** *Let  $G = (V, E)$  be a connected and undirected graph,  $S \subseteq V$  such that  $G[S]$  is connected, and  $W_S$  be the sub-matrix of  $W$  consisting of rows and columns indexed by the vertices in  $S$ . Then, it holds for any integer  $t > 0$  that*

$$\mathbb{E}_{u \sim \mathbf{p}_S} [\text{rem}(u, t, S)] = \left(1 - \frac{1 - \lambda_S}{2}\right)^t = \lambda_{\max}(W_S)^t, \tag{3}$$

where  $\mathbf{p}_S \triangleq D_S \mathbf{z}_S / \|D_S \mathbf{z}_S\|_1$  and  $\mathbf{z}_S$  is the eigenvector with respect to  $\lambda_{\max}(W_S)$ .

**Remark 1.** *Since  $W_S = (I_S + D_S^{-1}A_S)/2$  by definition, it is easy to check that  $[1 + \lambda_i(M_S)]/2 = \lambda_i(W_S)$  for all  $1 \leq i \leq |S|$ . Thus,  $(1 + \lambda_S)/2 = \lambda_{\max}(W_S)$ .*

Theorem 5 implies the existence of good vertices, i.e., vertices from which the random walk always stays in  $S$  with substantially higher probability.

**Corollary 1.** *Let  $G = (V, E)$  be a graph, and  $S \subseteq V$  a subset such that  $G[S]$  is connected. Then, for any integer  $t > 0$ , there is at least one vertex  $u \in S$  such that*

$$\text{rem}(u, t, S) \geq \left(1 - \frac{1 - \lambda_S}{2}\right)^t = \lambda_{\max}(W_S)^t.$$

**Lemma 2.** *Let  $G[S]$  be a connected graph. Then,  $\lambda_S = 1 - \phi_S$  if and only if the significance vector  $\mathbf{z}_S = \mathbf{1}/\sqrt{\text{vol}(S)}$  is a constant vector.*

Comparing with Corollary 1, all previous bounds on the staying probability are with respect to the conductance of  $S$  [10, 16]. Gharan and Trevisan proved that  $\mathbb{E}_{u \sim \pi_S} [\text{rem}(u, t, S)] \geq (1 - \phi_S/2)^t$  for any set  $S \subseteq V$ , where  $(\pi_S)_u = \text{deg}(u)/\text{vol}(S)$  if  $u \in S$ , and 0 otherwise. Their result implies the existence of a vertex  $u \in S$  such that  $\text{rem}(u, t, S) \geq (1 - \phi_S/2)^t$ . Due to Lemma 1 and Lemma 2, in almost all cases our result strictly improves the previous best. The main technique used in proving Theorem 5 is to analyze the Dirichlet eigenvalues and the corresponding eigenvectors of matrix  $M_S$ . We remark that Kwok and Lau [12] implicitly established an identical result to Theorem 5.

### 4 Local Mixing Time

For any random walk with transition matrix  $W$ , the stationary distribution  $\pi$  satisfy  $\pi^\top M = \pi^\top$ . For a lazy random walk on any undirected graph with the transition matrix  $M$ , the stationary distribution  $\pi$  exists and is determined by the eigenvector corresponding to the largest eigenvalue of matrix  $M$ . However, for any subset  $S$ , the random walk started in  $S$  may leave  $S$  with non-zero probability in each step. This fact implies the non-existence of stable distribution in the case of local random walks (w.r.t. set  $S$ ).

On the other hand, Corollary 1 shows that the spectral structure of the matrix  $W_S$  influences the staying probability of set  $S$ . This interesting fact motivates us to study the local mixing time with respect to Dirichlet eigenvalues and the corresponding eigenvectors.

Let  $G = (V, E)$  be a graph, and  $S \subseteq V$  a subset. We denote the *induced Markov chain* by  $\mathcal{M}_S(G[S], W_S)$ , where  $W_S = (I_S + D_S^{-1}A_S)/2$  is a non-stochastic matrix. The next theorem proves that the mixing rate of a local random walk is determined by the first and the second largest eigenvalues of  $W_S$ , and the eigenvector corresponding to the largest eigenvalue of  $W_S$ .

**Theorem 6.** *Let  $\mathcal{M}(G, W)$  be a finite, time-reversible, and ergodic Markov chain, and  $S \subset V$  such that  $G[S]$  is connected. Let  $\mathcal{M}_S(G[S], W_S)$  be the induced Markov chain. Then, for any  $\varepsilon > 0$  there is a step  $\text{MIX}_S(\varepsilon)$  such that it holds for any different vertices  $u, v \in S$  and step  $t \geq \text{MIX}_S(\varepsilon)$  that*

$$\left| (W_S^t)_{uv} - \alpha_u^{(t)} \cdot (\mathbf{p}_S)_v \right| \leq \varepsilon \cdot \alpha_u^{(t)} \cdot (\mathbf{p}_S)_v, \tag{4}$$

where each coordinate of  $\alpha^{(t)}$  is defined by

$$\alpha_u^{(t)} \triangleq (\mathbf{z}_S)_u \cdot \lambda_{\max}(W_S)^t \cdot \|D_S \mathbf{z}_S\|_1, \tag{5}$$

and  $\mathbf{z}_S$  is the eigenvector corresponding to  $\lambda_{\max}(W_S)$ . Moreover, the local mixing time is upper bounded by

$$\text{MIX}_S(\varepsilon) \leq \frac{\lambda_{\max}(W_S)}{\lambda_{\max}(W_S) - \lambda_2(W_S)} \cdot \ln \left( \frac{1}{\min_{u \in S} \left\{ \text{deg}(u) \cdot (\mathbf{z}_S)_u^2 \right\}} \cdot \frac{1}{\varepsilon} \right). \tag{6}$$

Let us first discuss the meaning of the local mixing time. For a  $t$ -step random walk with transition matrix  $W_S$  and starting vertex  $u \in S$ , the probability that the walk reaches vertex  $v$  is  $(W_S^t)_{u,v}$ . Theorem 6 states that after  $t \geq \text{MIX}_S(\varepsilon)$  steps, the induced *pseudo distribution*<sup>2</sup>  $\mathbf{1}_u^\top W_S^t$  becomes *almost stable*, in the sense that for  $\varepsilon' = 2\varepsilon/(1 - \varepsilon)$  and any vertices  $v, w \in S$  it holds that

$$(1 - \varepsilon') \cdot (\mathbf{p}_S)_v / (\mathbf{p}_S)_w \leq (W_S^t)_{uv} / (W_S^t)_{uw} \leq (1 + \varepsilon') \cdot (\mathbf{p}_S)_v / (\mathbf{p}_S)_w.$$

Hence, all induced pseudo distributions  $\{\mathbf{1}_u^\top W_S^t\}_{u \in S}$  have approximately the same shape as the distribution  $\mathbf{p}_S$ . This is the reason why we call  $\mathbf{p}_S$  the *almost stable distribution*.

*Proof (Proof of Theorem 6).* Our proof generalizes the approach proposed in [11]. Let  $|S| = k$ . We define by  $W_S^N \triangleq D_S^{1/2} W_S D_S^{-1/2} = (I_S + M_S) / 2$  a normalized and symmetric matrix. Observe that  $(W_S^N)^t = D_S^{1/2} (W_S)^t D_S^{-1/2}$  and hence  $W_S^t = D_S^{-1/2} (W_S^N)^t D_S^{1/2}$ .

Moreover, by Remark 1 and since  $W_S^N$  is symmetric,  $(W_S^N)^t$  has an eigen-decomposition of the form  $(W_S^N)^t = \sum_{i=1}^k \left(\frac{1+\lambda_i}{2}\right)^t \mathbf{v}_i \mathbf{v}_i^\top = \sum_{i=1}^k \lambda_i (W_S)^t \mathbf{v}_i \mathbf{v}_i^\top$ , where  $\mathbf{v}_i$  and  $\lambda_i$  are the  $i$ th eigenvector and eigenvalue of matrix  $M_S$  respectively.

We prove next the upper bound on  $\text{MIX}_S(\varepsilon)$  for any  $\varepsilon > 0$ . To simplify the notations we define by  $V_S^{(i)} \triangleq D_S^{-1/2} \mathbf{v}_i \mathbf{v}_i^\top D_S^{1/2}$  the scaled tensor product of eigenvector  $\mathbf{v}_i$ . Notice that

$$(W_S^t)_{uv} = \lambda_{\max}(W_S)^t \cdot (V_S^{(1)})_{uv} + \sum_{i=2}^k \lambda_i (W_S)^t \cdot (V_S^{(i)})_{uv}. \tag{7}$$

By definition, for every vertex  $u \in S$  it holds that

$$\lambda_{\max}(W_S)^t \cdot \mathbf{1}_u^\top V_S^{(1)} = \left[ (\mathbf{z}_S)_u \cdot \lambda_{\max}(W_S)^t \cdot \|D_S \mathbf{z}_S\|_1 \right] \cdot \mathbf{p}_S^\top, \tag{8}$$

Now we set  $\alpha_u^{(t)} \triangleq (\mathbf{z}_S)_u \cdot \lambda_{\max}(W_S)^t \cdot \|D_S \mathbf{z}_S\|_1$ . By (7), (8) and due to the orthogonality of eigenvectors  $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$  it follows that

$$\sum_{i=2}^k \left| (V_S^{(i)})_{uv} \right| = \frac{\sqrt{\deg(v)}}{\sqrt{\deg(u)}} \sum_{i=2}^k |(\mathbf{v}_i)_u| |(\mathbf{v}_i)_v| \leq \sqrt{\frac{\deg(v)}{\deg(u)}},$$

and hence

$$\left| (W_S^t)_{uv} - \alpha_u^{(t)} \cdot (\mathbf{p}_S)_v \right| = \left| \sum_{i=2}^k \lambda_i (W_S)^t \cdot (V_S^{(i)})_{uv} \right| \leq \lambda_2 (W_S)^t \cdot \sqrt{\frac{\deg(v)}{\deg(u)}}. \tag{9}$$

Furthermore, by definition of  $\mathbf{p}_S$ ,  $\mathbf{z}_S$  and  $\alpha_u^{(t)}$  it holds that

$$\frac{1}{\alpha_u^{(t)} \cdot (\mathbf{p}_S)_v} = \sqrt{\frac{\deg(u)}{\deg(v)}} \cdot \frac{1}{\mathbf{v}_u \mathbf{v}_v} \cdot \frac{1}{\lambda_{\max}(W_S)^t}. \tag{10}$$

<sup>2</sup> A pseudo distribution  $\mathbf{q} \in \mathbb{R}^S$  satisfies:  $\mathbf{1}^\top \mathbf{q} < 1$  and  $\mathbf{q}_i \geq 0$  for every  $i \in S$ .

By (9), (10) and  $\mathbf{z}_S = D_S^{-1/2} \mathbf{v}$  we have multiplicative approximation of  $W_S^t$

$$\frac{|(W_S^t)_{uv} - \alpha_u^{(t)} \cdot (\mathbf{P}_S)_v|}{\alpha_u^{(t)} \cdot (\mathbf{P}_S)_v} \leq \frac{1}{\min_{v \in S} \{ \deg(v) \cdot (\mathbf{z}_S)_v^2 \}} \cdot \left( \frac{\lambda_2(W_S)}{\lambda_{\max}(W_S)} \right)^t. \tag{11}$$

Straightforward checking shows that the right hand side of (11) is at most  $\varepsilon > 0$  for any  $t \geq \frac{\lambda_{\max}(W_S)}{\lambda_{\max}(W_S) + \lambda_2(W_S)} \cdot \ln \left( \frac{1}{\min_{v \in S} \{ \deg(v) \cdot (\mathbf{z}_S)_v^2 \}} \cdot \frac{1}{\varepsilon} \right)$ . This yields the desired upper bound on the local mixing time  $\text{MIX}_S(\varepsilon)$ . ■

We further remark that the notation of the local mixing time is a natural generalization of the well-studied uniform mixing time of Markov chains. To see this, we apply Theorem 6 with  $S = V$  and obtain that

$$\max_{u, v \in V} \left| \frac{(W^t)_{uv} - \pi_v}{\pi_v} \right| \leq \frac{\lambda_2(W)^t}{\min_{w \in V} \pi_w}, \tag{12}$$

which implies that  $\text{MIX}_S(\varepsilon) \leq \frac{1}{1 - \lambda_2(W)} \cdot \ln \left( \frac{\text{vol}(V)}{\min_{u \in S} \deg(u)} \cdot \frac{1}{\varepsilon} \right)$ .

### 5 $\varepsilon$ -Uniform Mixing Time

We further study  $\varepsilon$ -Uniform Mixing Time ( $\varepsilon$ -UMT) defined in (1). By definition,  $\tau(\varepsilon)$  is the minimum number of steps such that the largest relative difference between  $\mathcal{P}_u[X_t = v]$  and  $\pi_v$  is at most  $\varepsilon \cdot \pi_v$ , maximized over all initial vertices  $u, v \in V(G)$ . It was proven in [11] that the  $\varepsilon$ -UMT of any lazy random walk satisfies

$$\tau(\varepsilon) \leq \frac{2}{\phi_G^2} \log \left( \frac{1}{\min_{v \in V} \pi_v} \cdot \frac{1}{\varepsilon} \right).$$

Gharan and Trevisan [10] presented a lower bound on  $\tau(\varepsilon)$ , for any  $0 < \varepsilon < 1$ , by proving that

$$\tau(\varepsilon) \geq \frac{1}{2\phi_S} \ln \left( \frac{\text{vol}(V)}{2\gamma} \right), \tag{13}$$

where  $1 \leq \gamma \leq \text{vol}(V)/2$  and  $\phi_S \leq 0.7$ . Our result for the  $\varepsilon$ -UMT is as follows.

**Theorem 2** (from page 623). *Let  $G = (V, E)$  be an undirected and connected graph, and  $\mathcal{S}_\gamma \triangleq \{S \subseteq V \mid \text{vol}(S) \leq \gamma, G[S] \text{ is connected and } \lambda_S \geq 1/4\}$ . Then,*

$$\tau(\varepsilon) \geq \max_{1 \leq \gamma \leq \text{vol}(S)/2} \max_{S \in \mathcal{S}_\gamma} \left\{ \frac{1}{2(1 - \lambda_S)} \ln \left( \frac{\text{vol}(V)}{2\gamma} \right) \right\}. \tag{14}$$

Comparing with the previous best known result (13) that only depends on the conductance of subsets, the result in Theorem 2 is based on the Dirichlet

eigenvalue  $\lambda_S$ . Moreover, Theorem 2 provides an improved result over the state of the art, due to  $1 - \lambda_S \leq \phi_S$  (cf. Lemma 1).

Furthermore, since the value of  $\lambda_S$  depends on both the inner structure of  $G[S]$  and the conductance of  $S$ , Theorem 2 implies that in contrast to simply considering  $\phi_S$ , the inner structure of  $G[S]$  influences the  $\varepsilon$ -UMT. To the best of our knowledge, this is the first result relating the mixing time to the local structure of connected subsets of vertices in  $G$ .

## 6 An Improved Algorithm for Graph Partitioning

In this section we present and analyze the algorithm corresponding to Theorem 3.

We start with introducing one technique introduced by Lovasz and Simonovits [13, 14]. Let  $\mathbf{q}$  be a probability distribution over the vertices in  $G$  and  $\sigma : V \mapsto V$  a permutation of vertices such that it is non-increasing with respect to  $\mathbf{q}_u/\text{deg}(u)$ , where the ties are resolved lexicographically, i.e.

$$\frac{\mathbf{q}(\sigma(1))}{\text{deg}(\sigma(1))} \geq \dots \geq \frac{\mathbf{q}(\sigma(i))}{\text{deg}(\sigma(i))} \geq \dots \geq \frac{\mathbf{q}(\sigma(n))}{\text{deg}(\sigma(n))}.$$

We write  $T_i(\mathbf{q})$  to denote a *sweep set* of the first  $i$  vertices taken with respect to the permutation  $\sigma$ , i.e.,  $T_i(\mathbf{q}) \triangleq \{\sigma(1), \dots, \sigma(i)\}$ . For simplicity we write  $T_i$  instead of  $T_i(\mathbf{q})$  when  $\mathbf{q}$  is clear from the context.

Our algorithm follows the framework in [10], and finds a cluster with the worst case guarantee with respect to the Dirichlet eigenvalue  $\lambda_S$ , instead of conductance  $\phi_S$ . The formal description is as follows.

---

**Algorithm 1.** ASSEP( $\lambda, \varepsilon$ )

---

For all vertices  $v \in V$  and all integers  $1 \leq i \leq n - 1$  and  $1 \leq t \leq \frac{\varepsilon \cdot \ln \text{vol}(V)}{1 - \lambda}$ ,  
 let  $\mathcal{T}$  be the family of all connected sweep sets  $T_i(\mathbf{1}_v^\top W^t)$  satisfying the inequality  

$$1 - \lambda_{\max}(M_{T_i}) \leq \sqrt{2(1 - \lambda)}/\varepsilon$$

**return** the set with minimum volume in  $\mathcal{T}$ .

---

Now we analyze Algorithm 1. We first state a useful lemma that will be used in proving Theorem 3.

**Lemma 3** ([10]). *For any vertex  $v \in V$ ,  $0 \leq \gamma \leq m$ ,  $t \geq 0$  and  $0 \leq \Phi \leq 1/2$ , if for all  $t \leq T$ , all sweep sets  $T_i(\mathbf{1}_v^\top W^t)$  of volume at most  $\gamma$  have conductance at least  $\Phi$ , then it holds for any  $0 \leq t \leq T$  that*

$$C(\mathbf{1}_v^\top W^t, x) \leq \frac{x}{\gamma} + \sqrt{\frac{x}{\text{deg}(v)}} \left(1 - \frac{\Phi^2}{2}\right)^t.$$

*Proof of Theorem 3.* We follow the proof in [10], but in contrast apply Corollary 1 to obtain an improved lower bound on the staying probability expressed in terms of Dirichlet eigenvalue  $\lambda_S$ . We define by

$$\gamma = \text{vol}(U), \quad T = \varepsilon/(1 - \lambda) \cdot \ln \gamma, \quad \Gamma = 2\gamma^{1+\varepsilon}, \quad \text{and} \quad \Phi = \sqrt{2(1 - \lambda)}/\varepsilon.$$

We show that the algorithm *ASSEP*( $\lambda, \varepsilon$ ) returns a set  $S$  of volume at most  $\Gamma$  with the Dirichlet eigenvalue  $\lambda_S$  at least  $1 - \sqrt{2(1 - \lambda)}/\varepsilon$ . The proof is by contradiction.

Assume for contradiction that the output set  $S$  has volume larger than  $\Gamma$ . We show that Lemma 3 and Corollary 1 can not hold simultaneously. By Corollary 1, there is a vertex  $u \in U$ , such that

$$\text{rem}(u, t, U) \geq \left(1 - \frac{1 - \lambda_{\max}(M_U)}{2}\right)^t \geq \left(1 - \frac{1 - \lambda}{2}\right)^{\frac{\varepsilon}{1-\lambda} \ln \gamma} \geq \gamma^{-\varepsilon}. \quad (15)$$

We denote by  $\mathbf{q} = \mathbf{1}_u^\top W^t$  and construct a vector  $\mathbf{w} \in \mathbb{R}^V$  such that  $\mathbf{w}(v) = 1$ , for all  $v \in U$  and  $\mathbf{w}(v) = 0$ , otherwise. By the definition of Lovasz-Simonovits Curve [13, 14] and (15), we obtain that

$$C(\mathbf{q}, \gamma) \geq \sum_{v \in V} \mathbf{w}(v) \cdot \mathbf{q}_v = \sum_{v \in U} \mathbf{q}_v \geq \text{rem}(u, t, U) \geq \gamma^{-\varepsilon}. \quad (16)$$

By the assumption and the description of algorithm *ASSEP*( $\lambda, \varepsilon$ ), it follows that the minimum volume over all connected sweep sets with  $1 - \lambda_{\max}(M_{T_i}) \leq \sqrt{2(1 - \lambda)}/\varepsilon$  is at least  $\Gamma$ . Thus, each connected sweep set  $T_i$  of volume at most  $\Gamma$  satisfies the inequality  $1 - \lambda_{\max}(M_{T_i}) \geq \sqrt{2(1 - \lambda)}/\varepsilon$ . Furthermore, by Lemma 1 it follows that

$$\Phi_{T_i} \geq 1 - \lambda_{\max}(M_{T_i}) \geq \sqrt{2(1 - \lambda)}/\varepsilon. \quad (17)$$

The desired contradiction is obtained by combining (16), (17) and Lemma 3, since

$$\begin{aligned} C(\mathbf{q}, \gamma) &\leq \frac{\gamma}{\Gamma} + \sqrt{\frac{\gamma}{\deg(v)}} \left(1 - \frac{\left(\sqrt{2(1 - \lambda)}/\varepsilon\right)^2}{2}\right)^t \\ &\leq \frac{1}{2\gamma^\varepsilon} + \sqrt{\frac{\gamma}{\deg(v)}} \left(1 - \frac{1 - \lambda}{\varepsilon}\right)^{\frac{\varepsilon}{1-\lambda} \ln \gamma} < \gamma^{-\varepsilon}, \end{aligned}$$

where the first inequality uses the monotonicity of  $C(\mathbf{q}, \cdot)$  and the fact that  $x = \text{vol}(U) \leq \gamma$ , and the last inequality follows by  $\varepsilon < 1/2$  and  $\deg(v) \geq 4$ . ■

## 7 The Vertex Significance Ordering

In real networks a random walk starting from different vertices stays entirely in a cluster  $S$  with different probabilities, i.e., random walks initialized from the



boundary vertices leave  $S$  with higher probability, while random walks starting from the center vertices leave  $S$  with lower probabilities. This suggests a natural ordering of vertices in a cluster according to the staying probability of a  $t$ -step random walk.

From a theoretical point of view, random walks starting from vertices of higher significance have better chance to explore the whole cluster, and for some practical applications the staying probability of a vertex in  $t$  steps presents its significance in a cluster. Using probabilistic methods [16] showed that there is a subset of good vertices, and random walks starting from these vertices have provably higher chance to stay in  $S$ .

We further relate the significance of vertices in  $S$  to their associated staying probability. Recall that by Corollary 1, for any step  $t$  there is a vertex  $u$  such that  $\text{rem}(u, t, S) \geq \lambda_{\max}(W_S)^t$ . Our result shows that for any  $t \geq \text{MIX}_S(\varepsilon)$  the staying probability  $\text{rem}(u^*, t, S)$  of the most significant vertex  $u^* \in S$  can be lower bounded by the expected staying probability  $\mathbb{E}_{u \sim \mathbf{p}_S}[\text{rem}(u, t, S)] = \lambda_{\max}(W_S)^t$  (cf. to Theorem 5) up to a  $(1 - \varepsilon)$  factor.

**Theorem 7.** *Let  $\mathcal{M}(G = (V, E), W)$  be a finite, time-reversible, ergodic Markov chain,  $S \subset V$  such that  $G[S]$  is connected, and  $\mathcal{M}_S(G[S], W_S)$  an induced Markov chain. Then, for all  $\varepsilon > 0$ ,  $t \geq \text{MIX}_S(\varepsilon)$  and any vertex  $u \in S$ , it holds that*

$$\{(1 - \varepsilon) \xi_t\}(\mathbf{z}_S)_u \leq \text{rem}(u, t, S) \leq \{(1 + \varepsilon) \xi^t\}(\mathbf{z}_S)_u,$$

where  $\xi_t = \lambda_{\max}(W_S)^t \cdot \|D_S \mathbf{z}_S\|_1$ . Furthermore, the most significant vertex  $u^*$  satisfies

$$\text{rem}(u^*, t, S) \geq (1 - \varepsilon) \cdot \lambda_{\max}(W_S)^t.$$

Now we present an efficient algorithm that orders the vertices in  $S$  according to their significance. Our algorithm is based on the analysis of local random walks in Section 4, and its description is as follows.

---

**Algorithm 2.** Vertex Significance Ordering (VSO)

---

**Require:** Graph  $G = (V, E)$  with the adjacency matrix  $A$  and subset  $S \subseteq V$  of  $|S| = k$  vertices such that  $G[S]$  is connected.

- 1: Compute the vector  $\mathbf{z}_S = D_S^{-1/2} \mathbf{v}$ , where  $\mathbf{v} \in \mathbb{R}_{>0}^S$  is the normalized eigenvector of matrix  $M_S = D_S^{-1/2} A_S D_S^{-1/2}$  with corresponding eigenvalue  $\lambda_S$ .
  - 2: Sort all entries of  $\mathbf{z}_S$  in non-increasing order and resolve ties lexicographically.
  - 3: **return** the index set  $\{\tau_1, \dots, \tau_k\}$  such that  $(\mathbf{z}_S)_{\tau_1} \geq \dots \geq (\mathbf{z}_S)_{\tau_k}$ .
- 

As a straightforward application of Theorem 7, Algorithm 2 and Theorem 5 we obtain the following result.

**Corollary 2.** *Let  $G = (V, E)$  be a graph,  $S \subset V$  a subset such that  $G[S]$  is connected, and  $\varepsilon > 0$  an accuracy parameter. Then, given the set  $S$ , Algorithm 2 efficiently finds a vertex  $u^* \in S$  such that for any  $t \geq \text{MIX}_S(\varepsilon)$  it holds that*

$$\text{rem}(u^*, t, S) \geq (1 - \varepsilon) \cdot \lambda_{\max}(W_S)^t = (1 - \varepsilon) \cdot \mathbb{E}_{u \sim \mathbf{p}_S}[\text{rem}(u, t, S)].$$

## References

1. Andersen, R., Chung, F.R.K., Lang, K.J.: Using pagerank to locally partition a graph. *Internet Mathematics* **4**(1), 35–64 (2007)
2. Andersen, R., Peres, Y.: Finding sparse cuts locally using evolving sets. In: 41st Annual Symposium on Theory of Computing, STOC 2009, pp. 235–244 (2009)
3. Berenbrink, P., Cooper, C., Elsässer, R., Radzik, T., Sauerwald, T.: Speeding up random walks with neighborhood exploration. In: 21st Annual Symposium on Discrete Algorithms, SODA 2010, pp. 1422–1435 (2010)
4. Chung, F.R.K.: *Spectral Graph Theory*. American Mathematical Society (1997)
5. Chung, F.: Random walks and local cuts in graphs. *Linear Algebra and its Applications* **423**(1), 22–32 (2007)
6. Chung, F., Yau, S.-T.: Discrete green’s functions. *J. Comb. Theory Series A* **91**(1–2), 214 (2000)
7. Cooper, C., Elsässer, R., Ono, H., Radzik, T.: Coalescing random walks and voting on connected graphs. *SIAM J. Discrete Math.* **27**(4), 1748–1758 (2013)
8. Elsässer, R., Sauerwald, T.: Tight bounds for the cover time of multiple random walks. *Theor. Comput. Sci.* **412**(24), 2623–2641 (2011)
9. Friedrich, T., Sauerwald, T.: The cover time of deterministic random walks. *Electr. J. Comb.* **17**(1) (2010)
10. Gharan, S.O., Trevisan, L.: Approximating the expansion profile and almost optimal local graph clustering. In: 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012, pp. 187–196 (2012)
11. Jerrum, M., Sinclair, A.: Approximating the permanent. *SIAM J. Comput.* **18**(6), 1149–1178 (1989)
12. Gharan, S.O., Trevisan, L.: Approximating the expansion profile and almost optimal local graph clustering. In: 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012, pp. 187–196 (2012)
13. Lovász, L., Simonovits, M.: The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In: 31st Annual Symposium on Foundations of Computer Science, FOCS 1990, pp. 346–354 (1990)
14. Lovász, L., Simonovits, M.: Random walks in a convex body and an improved volume algorithm. *Rand. Struct. & Algo.* **4**(4), 359–412 (1993)
15. Sauerwald, T., Sun, H.: Tight bounds for randomized load balancing on arbitrary network topologies. In: 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012, pp. 341–350 (2012)
16. Spielman, D.A., Teng, S.-H.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: 36th Annual Symposium on Theory of Computing, STOC 2004, pp. 81–90 (2004)
17. Zhu, Z.A., Lattanzi, S., Mirrokni, V.S.: A local algorithm for finding well-connected clusters. In: 30th International Conference on Machine Learning, ICML 2013, pp. 396–404 (2013)

# Planar Embeddings with Small and Uniform Faces

Giordano Da Lozzo<sup>1</sup>(✉), Vít Jelínek<sup>2</sup>, Jan Kratochvíl<sup>3</sup>,  
and Ignaz Rutter<sup>3,4</sup>

<sup>1</sup> Department of Engineering, Roma Tre University, Rome, Italy  
`dalozzo@dia.uniroma3.it`

<sup>2</sup> Computer Science Institute, Faculty of Mathematics and Physics,  
Charles University, Prague, Czech Republic  
`jelinek@iuuk.mff.cuni.cz`

<sup>3</sup> Department of Applied Mathematics, Faculty of Mathematics and Physics,  
Charles University, Prague, Czech Republic  
`honza@kam.mff.cuni.cz`

<sup>4</sup> Faculty of Informatics, Karlsruhe Institute of Technology (KIT),  
Karlsruhe, Germany  
`rutter@kit.edu`

**Abstract.** Motivated by finding planar embeddings that lead to drawings with favorable aesthetics, we study the problems MINMAXFACE and UNIFORMFACES of embedding a given biconnected multi-graph such that the largest face is as small as possible and such that all faces have the same size, respectively. We prove a complexity dichotomy for MINMAXFACE and show that deciding whether the maximum is at most  $k$  is polynomial-time solvable for  $k \leq 4$  and NP-complete for  $k \geq 5$ . Further, we give a 6-approximation for minimizing the maximum face in a planar embedding. For UNIFORMFACES, we show that the problem is NP-complete for odd  $k \geq 7$  and even  $k \geq 10$ . Moreover, we characterize the biconnected planar multi-graphs admitting 3- and 4-uniform embeddings (in a  $k$ -uniform embedding all faces have size  $k$ ) and give an efficient algorithm for testing the existence of a 6-uniform embedding.

## 1 Introduction

While there are infinitely many ways to embed a connected planar graph into the plane without edge crossings, these embeddings can be grouped into a finite number of equivalence classes, so-called *combinatorial embeddings*, where two embeddings are *equivalent* if the clockwise order around each vertex is the same. Many algorithms for drawing planar graphs require that the input graph is provided

---

Work by Giordano Da Lozzo was supported in part by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project “AMANDA – Algorithmics for MAssive and Networked DATA”. Work by Jan Kratochvíl and Vít Jelínek was supported by the grant no. 14-14179S of the Czech Science Foundation GAČR. Ignaz Rutter was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

together with a combinatorial embedding, which the algorithm preserves. Since the aesthetic properties of the drawing often depend critically on the chosen embedding, e.g. the number of bends in orthogonal drawings, it is natural to ask for a planar embedding that will lead to the best results.

In many cases the problem of optimizing some cost function over all combinatorial embeddings is NP-complete. For example, it is known that it is NP-complete to test the existence of an embedding that admits an orthogonal drawing without bends or an upward planar embedding [10]. On the other hand, there are efficient algorithms for minimizing various measures such as the radius of the dual [1,2] and attempts to minimize the number of bends in orthogonal drawings subject to some restrictions [3,4,6].

Usually, choosing a planar embedding is considered as deciding the circular ordering of edges around vertices. It can, however, also be equivalently viewed as choosing the set of facial cycles, i.e., which cycles become face boundaries. In this sense it is natural to seek an embedding whose facial cycles have favorable properties. For example, Gutwenger and Mutzel [12] give algorithms for computing an embedding that maximizes the size of the outer face. The most general form of this problem is as follows. The input consists of a graph and a cost function on the cycles of the graph, and we seek a planar embedding where the sum of the costs of the facial cycles is minimum. This general version of the problem has been introduced and studied by Mutzel and Weiskircher [14]. Woeginger [15] shows that it is NP-complete even when assigning cost 0 to all cycles of size up to  $k$  and cost 1 for longer cycles. Mutzel and Weiskircher [14] propose an ILP formulation for this problem based on SPQR-trees.

In this paper, we focus on two specific problems of this type, aimed at reducing the visual complexity and eliminating certain artifacts related to face sizes from drawings. Namely, large faces in the interior of a drawing may be perceived as holes and consequently interpreted as an artifact of the graph. Similarly, if the graph has faces of vastly different sizes, this may leave the impression that the drawn graph is highly irregular. However, rather than being a structural property of the graph, it is quite possible that the artifacts in the drawing rather stem from a poor embedding choice and can be avoided by choosing a more suitable planar embedding.

We thus propose two problems. First, to avoid large faces in the drawing, we seek to minimize the size of the largest face; we call this problem MINMAXFACE. Second, we study the problem of recognizing those graphs that admit perfectly uniform face sizes; we call this problem UNIFORMFACES. Both problems can be solved by the ILP approach of Mutzel and Weiskircher [14] but were not known to be NP-hard.

*Our Contributions.* First, in Section 3, we study the computational complexity of MINMAXFACE and its decision version  $k$ -MINMAXFACE, which asks whether the input graph can be embedded such that the maximum face size is at most  $k$ . We prove a complexity dichotomy for this problem and show that  $k$ -MINMAXFACE is polynomial-time solvable for  $k \leq 4$  and NP-complete for  $k \geq 5$ . Our hardness result for  $k \geq 5$  strengthens Woeginger's result [15], which states that

it is NP-complete to minimize the number of faces of size greater than  $k$  for  $k \geq 4$ , whereas our reduction shows that it is in fact NP-complete to decide whether such faces can be completely avoided. Furthermore, we give an efficient 6-approximation for MINMAXFACE.

Second, in Section 4, we study the problem of recognizing graphs that admit perfectly uniform face sizes (UNIFORMFACES), which is a special case of  $k$ -MINMAXFACE. An embedding is  $k$ -uniform if all faces have size  $k$ . We characterize the biconnected multi-graphs admitting a  $k$ -uniform embedding for  $k = 3, 4$  and give an efficient recognition algorithm for  $k = 6$ . Finally, we show that for odd  $k \geq 7$  and even  $k \geq 10$ , it is NP-complete to decide whether a planar graph admits a  $k$ -uniform embedding.

For space limitations, proofs are sketched or omitted; refer to [5] for full proofs.

## 2 Preliminaries

A graph  $G = (V, E)$  is *connected* if there is a path between any two vertices. A *cutvertex* is a vertex whose removal disconnects the graph. A separating pair  $\{u, v\}$  is a pair of vertices whose removal disconnects the graph. A connected graph is *biconnected* if it does not have a cutvertex and a biconnected graph is *3-connected* if it does not have a separating pair. Unless specified otherwise, throughout the rest of the paper we will consider graphs without loops, but with possible multiple edges.

We consider  $st$ -graphs with two special *pole* vertices  $s$  and  $t$ . The family of  $st$ -graphs can be constructed in a fashion very similar to series-parallel graphs. Namely, an edge  $st$  is an  $st$ -graph with poles  $s$  and  $t$ . Now let  $G_i$  be an  $st$ -graph with poles  $s_i, t_i$  for  $i = 1, \dots, k$  and let  $H$  be a planar graph with two designated adjacent vertices  $s$  and  $t$  and  $k + 1$  edges  $st, e_1, \dots, e_k$ . We call  $H$  the *skeleton* of the composition and its edges are called *virtual edges*; the edge  $st$  is the *parent edge* and  $s$  and  $t$  are the poles of the skeleton  $H$ . To compose the  $G_i$ 's into an  $st$ -graph with poles  $s$  and  $t$ , we remove the edge  $st$  from  $H$  and replace each  $e_i$  by  $G_i$  for  $i = 1, \dots, k$  by removing  $e_i$  and identifying the poles of  $G_i$  with the endpoints of  $e_i$ . In fact, we only allow three types of compositions: in a *series composition* the skeleton  $H$  is a cycle of length  $k + 1$ , in a *parallel composition*  $H$  consists of two vertices connected by  $k + 1$  parallel edges, and in a *rigid composition*  $H$  is 3-connected.

For every biconnected planar graph  $G$  with an edge  $st$ , the graph  $G - st$  is an  $st$ -graph with poles  $s$  and  $t$  [7]. Much in the same way as series-parallel graphs, the  $st$ -graph  $G - st$  gives rise to a (de-)composition tree  $\mathcal{T}$  describing how it can be obtained from single edges. The nodes of  $\mathcal{T}$  corresponding to edges, series, parallel, and rigid compositions of the graph are  $Q$ -,  $S$ -,  $P$ -, and  $R$ -nodes, respectively. To obtain a composition tree for  $G$ , we add an additional root  $Q$ -node representing the edge  $st$ . We associate with each node  $\mu$  the skeleton of the composition and denote it by  $\text{skel}(\mu)$ . For a  $Q$ -node  $\mu$ , the skeleton consists of the two endpoints of the edge represented by  $\mu$  and one real and one virtual

edge between them representing the rest of the graph. For a node  $\mu$  of  $\mathcal{T}$ , the *pertinent graph*  $\text{pert}(\mu)$  is the subgraph represented by the subtree with root  $\mu$ . For a virtual edge  $\varepsilon$  of a skeleton  $\text{skel}(\mu)$ , the *expansion graph* of  $\varepsilon$  is the pertinent graph  $\text{pert}(\mu')$  of the neighbor  $\mu'$  corresponding to  $\varepsilon$  when considering  $\mathcal{T}$  rooted at  $\mu$ .

The *SPQR-tree* of  $G$  with respect to the edge  $st$ , originally introduced by Di Battista and Tamassia [7], is the (unique) smallest decomposition tree  $\mathcal{T}$  for  $G$ . Using a different edge  $s't'$  of  $G$  and a composition of  $G - s't'$  corresponds to rerooting  $\mathcal{T}$  at the node representing  $s't'$ . It thus makes sense to say that  $\mathcal{T}$  is the SPQR-tree of  $G$ . The SPQR-tree of  $G$  has size linear in  $G$  and can be computed in linear time [11]. Planar embeddings of  $G$  correspond bijectively to planar embeddings of all skeletons of  $\mathcal{T}$ ; the choices are the orderings of the parallel edges in P-nodes and the embeddings of the R-node skeletons, which are unique up to a flip. When considering rooted SPQR-trees, we assume that the embedding of  $G$  is such that the root edge is incident to the outer face, which is equivalent to the parent edge being incident to the outer face in each skeleton. We remark that in a planar embedding of  $G$ , the poles of any node  $\mu$  of  $\mathcal{T}$  are incident to the outer face of  $\text{pert}(\mu)$ . Hence, in the following we only consider embeddings of the pertinent graphs with their poles lying on the same face.

### 3 Minimizing the Maximum Face

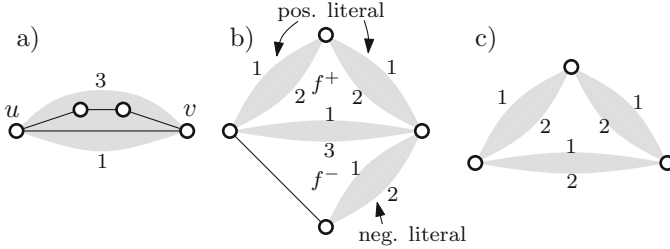
In this section we present our results on MINMAXFACE. We first strengthen the result of Woeginger [15] and show that  $k$ -MINMAXFACE is NP-complete for  $k \geq 5$  and then present efficient algorithms for  $k = 3, 4$ . In particular, the hardness result also implies that the problem MINMAXFACE is NP-hard. Finally, we give an efficient 6-approximation for MINMAXFACE on biconnected graphs. Recall that we allow graphs to have multiple edges.

**Theorem 1.**  *$k$ -MINMAXFACE is NP-complete for any  $k \geq 5$ .*

*Sketch of Proof.* Clearly, the problem is in NP. We sketch hardness for  $k = 5$ . Our reduction is from PLANAR 3-SAT where every variable occurs at most three times, which is NP-complete [8, Lemma 2.1]. Note that we can assume without loss of generality that for each variable both literals appear in the formula and, by replacing some variables by their negations, we can assume that a variable with three occurrences occurs twice as a positive literal. Let  $\varphi$  be such a formula.

We construct gadgets where some of the edges are in fact two parallel paths, one consisting of a single edge and one of length 2 or 3. The ordering of these paths then decides which of the faces incident to the gadget edge is incident to a path of length 1 and which is incident to a path of length 2 or 3; see Fig. 1a. Due to this use, we also call these gadgets (1, 2)- and (1, 3)-edges, respectively.

The gadget for a variable with three occurrences is shown in Fig. 1b. The central (1, 3)-edge (*variable edge*) decides the truth value of the variable. Depending on its flip either the positive literal edges or the negative literal edge must be



**Fig. 1.** Illustration of the gadgets for the proof of Theorem 1. (a) A (1,3)-edge. (b) A variable gadget for a variable that occurs twice as a positive literal and once as a negative literal. Changing the flip of the (1,3)-edge in the middle (variable edge) forces flipping the upper two literal edges. (c) A clause gadget for a clause of size 3.

embedded such that they have a path of length 2 in the outer face, which corresponds to a literal with truth value **false**. Figure 1c shows a clause gadget with three incident literal variables. Its inner face has size at most 5 if not all incident (1,2)-edges transmit value **false**. Clauses of size 2 and variables occurring only twice work similarly.

We now construct a graph  $G_\varphi$  by replacing in the plane variable–clause graph of  $\varphi$  each variable and each clause by a corresponding gadget and identifying (1,2)-edges that represent the same variable, taking into account the embedding of the variable–clause graph. Finally, we arbitrarily triangulate all faces that are not inner faces of gadgets. Then the only embedding choices are the flips of the (1,2)- and (1,3)-edges. We claim that  $\varphi$  is satisfiable if and only if  $G_\varphi$  has a planar embedding where every face has size at most 5.  $\square$

### 3.1 Polynomial-Time Algorithm for Small Faces

Next, we show that  $k$ -MINMAXFACE is polynomial-time solvable for  $k = 3, 4$ . Note that, if the input graph is simple, the problem for  $k = 3$  is solvable if and only if the input graph is maximal planar. A bit more work is necessary if we allow parallel edges.

Let  $G$  be a biconnected planar graph. We devise a dynamic program on the SPQR-tree  $\mathcal{T}$  of  $G$ . Let  $\mathcal{T}$  be rooted at an arbitrary Q-node and let  $\mu$  be a node of  $\mathcal{T}$ . We call the clockwise and counterclockwise paths connecting the poles of  $\mu$  along the outer face the *boundary paths* of  $\text{pert}(\mu)$ . We say that an embedding of  $\text{pert}(\mu)$  has type  $(a, b)$  if and only if all its inner faces have size at most  $k$  and its boundary paths have length  $a$  and  $b$ , respectively. Such an embedding is also called an  $(a, b)$ -embedding. We assume that  $a \leq b$ .

Clearly, each of the two boundary paths of  $\text{pert}(\mu)$  in an embedding  $\mathcal{E}_\mu$  of type  $(a, b)$  will be a proper subpath of the boundary of a face in any embedding of  $G$  where the embedding of  $\text{pert}(\mu)$  is  $\mathcal{E}_\mu$ . Hence, when seeking an embedding where all faces have size at most  $k$ , we are only interested in the embedding  $\mathcal{E}_\mu$  if  $1 \leq a \leq b \leq k - 1$ . We define a partial order on the embedding types by

$(a', b') \preceq (a, b)$  if and only if  $a' \leq a$  and  $b' \leq b$ . Replacing an  $(a, b)$ -embedding  $\mathcal{E}_\mu$  of  $\text{pert}(\mu)$  by (a reflection of) an  $(a', b')$ -embedding  $\mathcal{E}'_\mu$  with  $(a', b') \preceq (a, b)$  does not create faces of size more than  $k$ ; all inner faces of  $\mathcal{E}'_\mu$  have size at most  $k$  by assumption, and the only other faces affected are the two faces incident to the two boundary paths of  $\mathcal{E}'_\mu$ , whose length does not increase. We thus seek to compute for each node  $\mu$  the minimal pairs  $(a, b)$  for which it admits an  $(a, b)$ -embedding. We remark that  $\text{pert}(\mu)$  can admit an embedding of type  $(1, b)$  for some value of  $b$  only if  $\mu$  is either a P-node or a Q-node.

**Theorem 2.** *3-MINMAXFACE can be solved in linear time for biconnected graphs.*

We now deal with the case  $k = 4$ , which is similar but more complicated. The relevant types are  $(1, 1), (1, 2), (1, 3), (2, 2), (2, 3)$ , and  $(3, 3)$ . We note that precisely the two elements  $(2, 2)$  and  $(1, 3)$  are incomparable with respect to  $\preceq$ . Thus, it seems that, rather than computing only the single smallest type for which each pertinent graph admits an embedding, we are now forced to find all minimum pairs for which the pertinent graph admits a corresponding embedding. However, by the above observation, if a pertinent graph  $\text{pert}(\mu)$  admits a  $(1, 3)$ -embedding, then  $\mu$  must be a P-node. However, if the parent of  $\mu$  is an S-node or an R-node, then using a  $(1, 3)$ -embedding results in a face of size at least 5. Thus, such an embedding can only be used if the parent is the root Q-node. If there is the choice of a  $(2, 2)$ -embedding in this case, it can of course also be used at the root. Therefore, we can mostly ignore the  $(1, 3)$ -case and consider the linearly ordered embedding types  $(1, 1), (1, 2), (2, 2), (2, 3)$  and  $(3, 3)$ . The running time stems from the fact that, for an R-node, we need to find a matching between the virtual edges whose expansion graphs admit a  $(1, 2)$ -embedding and the incident triangular faces of the skeleton.

**Theorem 3.** *4-MINMAXFACE can be solved in  $O(n^{1.5})$  time for biconnected graphs.*

### 3.2 Approximation Algorithm

In this section, we present a constant-factor approximation algorithm for the problem of minimizing the largest face in an embedding of a biconnected graph  $G$ . We omit the correctness proofs and some of the technical details.

We again solve the problem by dynamic programming on the SPQR-tree of  $G$ .

Let  $G$  be a biconnected planar graph, and let  $\mathcal{T}$  be its SPQR-tree, rooted at an arbitrary Q-node. Let  $\mu$  be a node of  $\mathcal{T}$ . We also include the parent edge in the embedding of  $\text{skel}(\mu)$ , by drawing it in the outer face. In such an embedding, the two faces incident to the parent edge are called *the outer faces*; the remaining faces are *inner faces*.

Recall that an  $(a, b)$ -embedding of  $\text{pert}(\mu)$  is an embedding whose boundary paths have lengths  $a$  and  $b$ , where we always assume that  $a \leq b$ . We say that an  $(a, b)$ -embedding of  $\text{pert}(\mu)$  is *out-minimal* if for any  $(a', b')$ -embedding of



$\text{pert}(\mu)$ , we have  $a \leq a'$  and  $b \leq b'$ . Note that an out-minimal embedding need not exist; e.g.,  $\text{pert}(\mu)$  may admit a  $(2, 4)$ -embedding and a  $(3, 3)$ -embedding, but no  $(a, b)$ -embedding with  $a \leq 2$  and  $b \leq 3$ . We will later show, however, that such a situation can only occur when  $\mu$  is an S-node.

Let  $\text{OPT}(G)$  denote the smallest integer  $k$  such that  $G$  has an embedding whose every face has size at most  $k$ . For a node  $\mu$  of  $\mathcal{T}$ , we say that an embedding of  $\text{pert}(\mu)$  is *c-approximate*, if each inner face of the embedding has size at most  $c \cdot \text{OPT}(G)$ .

Call an embedding of  $\text{pert}(\mu)$  *neat* if it is out-minimal and 6-approximate. The main result of this section is the next proposition.

**Proposition 1.** *Let  $G$  be a biconnected planar graph with SPQR tree  $\mathcal{T}$ , rooted at an arbitrary  $Q$ -node. Then the pertinent graph of every  $Q$ -node,  $P$ -node or  $R$ -node of  $\mathcal{T}$  has a neat embedding, and this embedding may be computed in polynomial time.*

Since the pertinent graph of the root of  $\mathcal{T}$  is the whole graph  $G$ , the proposition implies a polynomial 6-approximation algorithm for minimizing the largest face.

Our proof of Proposition 1 is constructive. Fix a node  $\mu$  of  $\mathcal{T}$  which is not an S-node. We now describe an algorithm that computes a neat embedding of  $\text{pert}(\mu)$ , assuming that neat embeddings are available for the pertinent graphs of all the descendant nodes of  $\mu$  that are not S-nodes. We distinguish cases based on the type of the node  $\mu$ . We here only present the two difficult cases, when  $\mu$  is a P-node or an R-node.

*P-nodes.* Suppose that  $\mu$  is a P-node with  $k$  child nodes  $\mu_1, \dots, \mu_k$ , represented by  $k$  skeleton edges  $e_1, \dots, e_k$ . Let  $G_i$  be the expansion graph of  $e_i$ . We construct the *expanded skeleton*  $\text{skel}^*(\mu)$  as follows: if for some  $i$  the child node  $\mu_i$  is an S-node whose skeleton is a path of length  $m$ , replace the edge  $e_i$  by a path of length  $m$ , whose edges correspond in a natural way to the edges of  $\text{skel}(\mu_i)$ .

Every edge  $e'$  of the expanded skeleton corresponds to a node  $\mu'$  of  $\mathcal{T}$  which is a child or a grand-child of  $\mu$ . Moreover,  $\mu'$  is not an S-node, and we may thus assume that we have already computed a neat embedding for  $\text{pert}(\mu')$ . Note that  $\text{pert}(\mu')$  is the expansion graph of  $e'$ .

For each  $i \in \{1, \dots, k\}$  define  $\ell_i$  to be the smallest value such that  $G_i$  has an embedding with a boundary path of length  $\ell_i$ . We compute  $\ell_i$  as follows: if  $\mu_i$  is not an S-node, then we already know a neat  $(a_i, b_i)$ -embedding of  $G_i$ , and we may put  $\ell_i = a_i$ . If, on the other hand,  $\mu_i$  is an S-node, then let  $m$  be the number of edges in the path  $\text{skel}(\mu_i)$ , and let  $G_i^1, G_i^2, \dots, G_i^m$  be the expansion graphs of the edges of the path. For each  $G_i^j$ , we have already computed a neat  $(a_j, b_j)$ -embedding, so we may now put  $\ell_i = \sum_{j=1}^m a_j$ . Clearly, this value of  $\ell_i$  corresponds to the definition given above.

We now fix two distinct indices  $\alpha, \beta \in \{1, \dots, k\}$ , so that the values  $\ell_\alpha$  and  $\ell_\beta$  are as small as possible; formally,  $\ell_\alpha = \min\{\ell_i; i = 1, \dots, k\}$  and  $\ell_\beta = \min\{\ell_i; i = 1, \dots, k \text{ and } i \neq \alpha\}$ .

Let us fix an embedding of  $\text{skel}(\mu)$  in which  $e_\alpha$  and  $e_\beta$  are adjacent to the outer faces. We extend this embedding of  $\text{skel}(\mu)$  into an embedding of  $\text{pert}(\mu)$  by replacing each edge of  $\text{skel}^*(\mu)$  by a neat embedding of its expansion graph, in such a way that the two boundary paths have lengths  $\ell_\alpha$  and  $\ell_\beta$ . Let  $\mathcal{E}$  be the resulting  $(\ell_\alpha, \ell_\beta)$ -embedding of  $\text{pert}(\mu)$ . The embedding  $\mathcal{E}$  is neat (we omit the proof).

*R-nodes.* Suppose now that  $\mu$  is an R-node. As with P-nodes, we define the *expanded skeleton*  $\text{skel}^*(\mu)$  by replacing each edge of  $\text{skel}(\mu)$  corresponding to an S-node by a path of appropriate length. The graph  $\text{skel}^*(\mu)$  together with the parent edge forms a subdivision of a 3-connected graph. In particular, its embedding is determined uniquely up to a flip and a choice of outer face. Fix an embedding of  $\text{skel}^*(\mu)$  and the parent edge, so that the parent edge is on the outer face. Let  $f_1$  and  $f_2$  be the two faces incident to the parent edge of  $\mu$ .

Let  $e$  be an edge of  $\text{skel}^*(\mu)$ , let  $G_e$  be its expansion graph, and let  $\mathcal{E}_e$  be a neat  $(a, b)$ -embedding of  $G_e$ , for some  $a \leq b$ . The boundary path of  $\mathcal{E}_e$  of length  $a$  will be called *the short side* of  $\mathcal{E}_e$ , while the boundary path of length  $b$  will be *the long side*. If  $a = b$ , we choose the long side and short side arbitrarily.

Our goal is to extend the embedding of  $\text{skel}^*(\mu)$  into an embedding of  $\text{pert}(\mu)$  by replacing each edge  $e$  of  $\text{skel}^*(\mu)$  with a copy of  $\mathcal{E}_e$ . In doing so, we have to choose which of the two faces incident to  $e$  will be adjacent to the short side of  $\mathcal{E}_e$ .

First of all, if  $e$  is an edge of  $\text{skel}^*(\mu)$  incident to one of the outer faces  $f_1$  or  $f_2$ , we embed  $\mathcal{E}_e$  in such a way that its short side is adjacent to the outer face. Since  $f_1$  and  $f_2$  do not share an edge in  $\text{skel}^*(\mu)$ , such an embedding is always possible, and guarantees that the resulting embedding of  $\text{pert}(\mu)$  will be out-minimal.

It remains to determine the orientation of  $\mathcal{E}_e$  for the edges  $e$  that are not incident to the outer faces, in such a way that the largest face of the resulting embedding will be as small as possible. Rather than solving this task optimally, we formulate a linear programming relaxation, and then apply a rounding step which will guarantee a constant factor approximation.

Intuitively, the linear program works as follows: given an edge  $e$  incident to a pair of faces  $f$  and  $g$ , and a corresponding graph  $G_e$  with a short side of length  $a$  and a long side of length  $b$ , rather than assigning the short side to one face and the long side to the other, we assign to each of the two faces a fractional value in the interval  $[a, b]$ , so that the two values assigned by  $e$  to  $f$  and  $g$  have sum  $a + b$ , and the maximum total amount assigned to a single face of  $\text{skel}^*(\mu)$  from its incident edges is as small as possible.

More precisely, we consider the linear program with the set of variables

$$\{M\} \cup \{x_{e,f}; e \text{ is an edge adjacent to face } f\},$$

where the goal is to minimize  $M$  subject to the following constraints:

- For every edge  $e$  adjacent to a pair of faces  $f$  and  $g$ , we have the constraints  $x_{e,f} + x_{e,g} = a + b$ ,  $a \leq x_{e,f} \leq b$  and  $a \leq x_{e,g} \leq b$ , where  $a \leq b$  are the lengths of the two boundary paths of  $\mathcal{E}_e$ .

- Moreover, if an edge  $e$  is adjacent to an outer face  $f \in \{f_1, f_2\}$  as well as an inner face  $g$ , then we set  $x_{e,f} = a$  and  $x_{e,g} = b$ , with  $a$  and  $b$  as above.
- For every inner face  $f$  of  $\text{skel}^*(\mu)$ , we have the constraint  $\sum_e x_{e,f} \leq M$ , where the sum is over all edges incident to  $f$ .

Given an optimal solution of the above linear program, we determine the embedding of  $\text{pert}(\mu)$  as follows: for an edge  $e$  of  $\text{skel}^*(\mu)$  incident to two inner faces  $f$  and  $g$ , if  $x_{e,f} \leq x_{e,g}$ , embed  $\mathcal{E}_e$  with its short side incident to  $f$  and long side incident to  $g$ . Let  $\mathcal{E}_\mu$  be the resulting embedding. It can be shown that  $\mathcal{E}_\mu$  is neat.

Proposition 1 yields a 6-approximation algorithm for the minimization of largest face in biconnected graphs.

**Theorem 4.** *A 6-approximation for MINMAXFACE in biconnected graphs can be computed in polynomial time.*

## 4 Perfectly Uniform Face Sizes

In this section we study the problem of deciding whether a biconnected planar graph admits a  $k$ -uniform embedding. Note that, due to Euler’s formula, a connected planar graph with  $n$  vertices and  $m$  edges has  $f = m - n + 2$  faces. In order to admit an embedding where every face has size  $k$ , it is necessary that  $2m = fk$ . Hence there is at most one value of  $k$  for which the graph may admit a  $k$ -uniform embedding.

In the following, we characterize the graphs admitting 3-uniform and 4-uniform embeddings, and we give an efficient algorithm for testing whether a graph admits a 6-uniform embedding. Finally, we show that testing whether a graph admits a  $k$ -uniform embedding is NP-complete for odd  $k \geq 7$  and even  $k \geq 10$ . We leave open the cases  $k = 5$  and  $k = 8$ .

Our characterizations and our testing algorithm use the recursive structure of the SPQR-tree. To this end, it is necessary to consider embeddings of pertinent graphs, where we only require that the interior faces have size  $k$ , whereas the outer face may have different size, although it must not be too large. We call such an embedding *almost  $k$ -uniform*. The following lemma states that the size of the outer face in such an embedding depends only on the number of vertices and edges in the pertinent graph.

**Lemma 1.** *Let  $G$  be a graph with  $n$  vertices and  $m$  edges with an almost  $k$ -uniform embedding. Then the outer face has length  $\ell = k(n - m - 1) + 2m$ .*

Thus, for small values of  $k$ , where the two boundary paths of the pertinent graph may have only few different lengths, the type of an almost  $k$ -uniform embedding (as defined in Section 4) is essentially fixed. Using this fact, the biconnected graphs (with possible multiple edges) admitting a  $k$ -uniform dual for  $k = 3, 4$  can be characterized. We remark that the simple graphs admitting 3-uniform and 4-uniform embeddings are precisely the maximal planar graphs and the maximal planar bipartite graphs.

**Theorem 5.** *A biconnected planar graph  $G$  admits 3-uniform embedding if and only if its SPQR-tree satisfies all of the following conditions.*

- (i) *S- and R-nodes are only adjacent to Q- and P-nodes.*
- (ii) *Every R-node skeleton is a planar triangulation.*
- (iii) *Every S-node skeleton has size 3.*
- (iv) *Every P-node with  $k$  neighbors has  $k$  even and precisely  $k/2$  of the neighbors are Q-nodes.*

**Theorem 6.** *A biconnected planar graph admits a 4-regular dual if and only if it is bipartite and satisfies the following conditions.*

- (i) *For each P-node either all expansion graphs satisfy  $m_e = 2n_e - 4$ , or half of them satisfy  $m_e = 2n_e - 5$  and the other half are Q-nodes.*
- (ii) *For each S- or R-node all faces have size 3 or 4; the expansion graphs of all edges incident to faces of size 4 satisfy  $m_e = 2n_e - 3$  and for each triangular face, there is precisely one edge whose expansion graph satisfies  $m_e = 2n_e - 4$ , the others satisfy  $m_e = 2n_e - 3$ .*

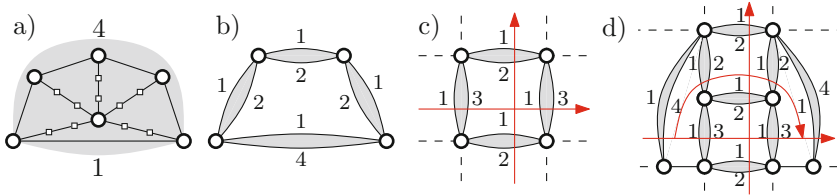
Theorems 5 and 6 can be used to construct linear-time algorithms to test whether a biconnected planar graph admits a 3-regular dual and a 4-regular dual, respectively.

**Theorem 7.** *It can be tested in  $O(n^{1.5})$  time whether a biconnected planar graph admits a 6-uniform embedding.*

*Sketch of Proof.* To test the existence of a 6-uniform embedding, we again use bottom-up traversal of the SPQR-tree and are therefore interested in the types of almost 6-uniform embeddings of pertinent graphs. Clearly, each of the two boundary paths of a pertinent graph may have length at most 5. Thus, only embeddings of type  $(a, b)$  with  $1 \leq a \leq b \leq 5$  are relevant. By Lemma 1 the value of  $a + b$  is fixed and in order to admit a  $k$ -uniform embedding with  $k$  even, it is necessary that the graph is bipartite. Thus, for an almost 6-uniform embedding the length of the outer face must be in  $\{2, 4, 6, 8, 10\}$ . Moreover, the color classes of the poles in the bipartite graph determine the parity of  $a$  and  $b$ .

For length 2 and length 10, the types must be  $(1, 1)$  and  $(5, 5)$ , respectively. For length 4, the type must be  $(1, 3)$  or  $(2, 2)$ , depending on the color classes of the poles. For length 6, the possible types are  $(2, 4)$  or  $(3, 3)$  (it can be argued that  $(1, 5)$  is not possible). Finally, for length 8, the possible types are  $(3, 5)$  and  $(4, 4)$  and again the color classes uniquely determine the type.

Thus, we know for each internal node  $\mu$  precisely what must be the type of an almost 6-uniform embedding of  $\text{pert}(\mu)$  if one exists. It remains to check whether for each node  $\mu$ , assuming that all children admit an almost 6-uniform embedding, it is possible to put them together to an almost 6-uniform embedding of  $\text{pert}(\mu)$ . For this, we need to decide (i) an embedding of  $\text{skel}(\mu)$  and (ii) for each child the flip of its almost  $k$ -uniform embedding. The main issue are R-nodes, where we have to solve a generalized matching problem to ensure that every face gets assigned a total boundary length of 6. This can be solved in  $O(n^{1.5})$  time [9]. □



**Fig. 2.** Illustration of the gadgets used for the hardness proof in Theorem 8. (a) A  $(1,4)$ -edge. (b) A variable gadget for a variable with three occurrences. (c), (d) Crossing gadgets for a pipe of  $(1,2)$ -edges with a pipe of  $(1,3)$ - and  $(1,4)$ -edges, respectively. The red arrows indicate the information flow.

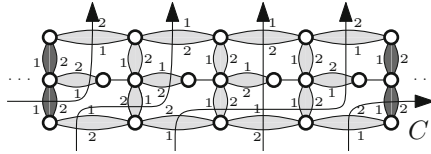
Finally, we prove NP-hardness for testing the existence of a  $k$ -uniform embedding for  $k = 7$  and  $k \geq 9$  by giving a reduction from the NP-complete problem PLANAR POSITIVE 1-IN-3-SAT where each variable occurs at least twice and at most three times and each clause has size two or three. The NP-completeness of this version of satisfiability follows from the results of Moore and Robson [13].

**Theorem 8.**  $k$ -UNIFORMFACES is NP-complete for all odd  $k \geq 7$  and even  $k \geq 10$ .

*Sketch of Proof.* We sketch the reduction for  $k = 7$ , the other cases are similar. We reduce from PLANAR POSITIVE 1-IN-3-SAT where each variable occurs two or three times and each clause has size two or three. Essentially, we perform a standard reduction, replacing each variable, each clause, and each edge of the variable–clause graph by a corresponding gadget, similar to the proof of Theorem 1.

First, it is possible to construct subgraphs that behave like  $(1,2)$ -,  $(1,3)$ -, and  $(1,4)$ -edges, i.e., their embedding is unique up to a flip, the inner faces have size 7 and the outer face has a path of length 1 and a path of length 2, 3 or 4 between the poles; see Fig. 2a for an example.

A variable is a cycle consisting of  $(1,2)$ -edges, called *output edges* (one for each occurrence of the variable) and one *sink-edge*, which is a  $(1,3)$ - or a  $(1,4)$ -edges depending on whether the variable occurs two or three times; see Fig. 2b. It is not hard to construct clauses with two or three incident  $(1,2)$ -edges whose internal face has size 7 if and only if exactly one of the incident  $(1,2)$ -edges contributes a path of length 1 to the internal face. We then use simple pipes to transmit the information encoded in the output edges to the clauses in a planar way. The main issue are the sink-edges, which have different length depending on whether the corresponding variable is **true** or **false**. To this end, we transfer the information encoded in all sink edges via pipes to a single face. We use the fact that sink-edges are  $(1,k)$ -edges with  $k > 2$  to cross over pipes transmitting these values using the crossing gadgets shown in Fig. 2c,d. Note that the construction in Fig. 2d is necessary since crossing a pipe of  $(1,4)$ -edges with a pipe of  $(1,2)$ -edges in the style of Fig. 2c would require face size at least 8.



**Fig. 3.** Illustration of a shift ring (the left and right dark gray edges are identified) that allows to transpose adjacent  $(1, 2)$ -edges encoding different states. The read arrows show the flow of information encoded in the  $(1, 2)$ -edges.

Now we have collected all the information encoded in the sink edges in a single face. By attaching variable gadgets to each of the corresponding pipes, we split this information into  $(1, 2)$ -edges, whose endpoints we identify such that they all form a single large cycle  $C$ .

Now, for all faces except for the inner faces of the gadgets and the face inside cycle  $C$ , we apply the following simple procedure. We triangulate them and insert into each triangle a new vertex connected to all its vertices by edges subdivided sufficiently often so that all faces have size 7. As a result the only remaining embedding choices are the flips of the  $(1, d)$ -edges used in the gadgets. We have that the original 1-in-3SAT formula is satisfiable if and only if the  $(1, d)$ -edges can be flipped so that all faces except the one inside  $C$  have size 7.

It follows from Lemma 1 that the length of the face inside  $C$  is uniquely determined if all other faces have size 7, but we do not know which of the  $(1, 2)$ -edges contribute paths of length 1 and which contribute paths of length 2. It then remains to give a construction that can subdivide the interior of  $C$  into faces of size 7 for any possible distribution. This is achieved by inserting ring-like structures that allow to shift and transpose adjacent edges that contribute paths of length 1 and length 2; see Fig. 3. By nesting sufficiently many such rings, we can ensure that in the innermost face the edges contributing paths of length 1 are consecutive, and the first one (in some orientation of  $C$ ) is at a fixed position. Then we can assume that we know exactly what the inner face looks like and we can use one of the previous constructions to subdivide it into faces of size 7. □

### 5 Conclusions and Open Problems

Throughout the paper we consider embeddings of planar multi-graphs on the sphere, that is, no face is regarded as the outer face. On the other hand, when dealing with embeddings in the plane, it seems natural to consider the variants of the MINMAXFACE and UNIFORMFACES problems in which the size of the outer face is not constrained. To simplify the analysis, we decided not to explicitly discuss these variants. However, both the hardness results and the embedding algorithms presented in the paper can be easily modified to handle outer faces of arbitrary size. In fact, in the reduction for  $k$ -MINMAXFACE and  $k$ -UNIFORMFACES, we can insert in any triangular face a cycle of arbitrary length and triangulate any but the

inner face bounded by the cycle. Also, when computing the type of an embedding of a node of the SPQR-tree, it is not difficult to additionally consider embeddings of the corresponding pertinent graph in which one of the two paths bounding the outer face has length greater than or equal to  $k$ .

We list some interesting open questions: What is the complexity of  $k$ -UNIFORMFACES for  $k = 5$  and  $k = 8$ ? Are UNIFORMFACES and MINMAXFACE polynomial-time solvable for biconnected series-parallel graphs? Are they FPT with respect to treewidth?

**Acknowledgments.** We thank Bartosz Walczak for discussions.

## References

1. Angelini, P., Di Battista, G., Patrignani, M.: Finding a minimum-depth embedding of a planar graph in  $O(n^4)$  time. *Algorithmica* **60**, 890–937 (2011)
2. Bienstock, D., Monma, C.L.: On the complexity of covering vertices by faces in a planar graph. *SIAM J. Comput.* **17**(1), 53–76 (1988)
3. Bläsius, T., Krug, M., Rutter, I., Wagner, D.: Orthogonal graph drawing with flexibility constraints. *Algorithmica* **68**, 859–885 (2014)
4. Bläsius, T., Rutter, I., Wagner, D.: Optimal orthogonal graph drawing with convex bend costs. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) *ICALP 2013, Part I. LNCS*, vol. 7965, pp. 184–195. Springer, Heidelberg (2013)
5. Da Lozzo, G., Jelínek, V., Kratochvíl, J., Rutter, I.: Planar Embeddings with Small and Uniform Faces. ArXiv e-prints (September 2014)
6. Di Battista, G., Liotta, G., Vargiu, F.: Spirality and optimal orthogonal drawings. *SIAM Journal on Computing* **27**(6), 1764–1811 (1998)
7. Di Battista, G., Tamassia, R.: On-line graph algorithms with SPQR-trees. In: Paterson, M. (ed.) *ICALP 1990. LNCS*, vol. 443, pp. 598–611. Springer, Heidelberg (1990)
8. Fellows, M.R., Kratochvíl, J., Middendorf, M., Pfeiffer, F.: The complexity of induced minors and related problems. *Algorithmica* **13**, 266–282 (1995)
9. Gabow, H.N.: An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In: *Theory of Computing, STOC 1983*, pp. 448–456. ACM (1983)
10. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. *SIAM J. on Comput.* **31**(2), 601–625 (2001)
11. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) *GD 2000. LNCS*, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
12. Gutwenger, C., Mutzel, P.: Graph embedding with minimum depth and maximum external face. In: Liotta, G. (ed.) *GD 2003. LNCS*, vol. 2912, pp. 259–272. Springer, Heidelberg (2004)
13. Moore, C., Robson, J.M.: Hard tiling problems with simple tiles. *Discrete Comput. Geom.* **26**(4), 573–590 (2001)
14. Mutzel, P., Weiskircher, R.: Optimizing over all combinatorial embeddings of a planar graph. In: Cornuéjols, G., Burkard, R.E., Woeginger, G.J. (eds.) *IPCO 1999. LNCS*, vol. 1610, pp. 361–376. Springer, Heidelberg (1999)
15. Woeginger, G.J.: Embeddings of planar graphs that minimize the number of long-face cycles. *Oper. Res. Lett.*, 167–168 (2002)

# Scheduling Unit Jobs with a Common Deadline to Minimize the Sum of Weighted Completion Times and Rejection Penalties

Nevzat Onur Domanic<sup>(✉)</sup> and C. Gregory Plaxton

Department of Computer Science, University of Texas at Austin, Texas, USA  
{onur,plaxton}@cs.utexas.edu

**Abstract.** We study the problem of scheduling unit jobs on a single machine with a common deadline where some jobs may be rejected. Each job has a weight and a profit and the objective is to minimize the sum of the weighted completion times of the scheduled jobs plus the sum of the profits of the rejected jobs. Our main result is an  $O(n \log n)$ -time algorithm for this problem. In addition, we show how to incorporate weighted tardiness penalties with respect to a common due date into the objective while preserving the  $O(n \log n)$  time bound. We also discuss connections to a special class of unit-demand auctions. Finally, we establish that certain natural variations of the scheduling problems that we study are NP-hard.

## 1 Introduction

In many scheduling problems, we are given a set of jobs, and our goal is to design a schedule for executing the entire set of jobs that optimizes a particular scheduling criterion. Scheduling with rejection, however, allows some jobs to be rejected, either to meet deadlines or to optimize the scheduling criterion, while possibly incurring penalties for rejected jobs. In this paper we study the problem of scheduling unit jobs (i.e., jobs with an execution requirement of one time unit) with individual weights ( $w_i$ ) and profits ( $e_i$ ) on a single machine with a common deadline ( $\bar{d}$ ) where some jobs may be rejected. If a job is scheduled by the deadline then its completion time is denoted by  $C_i$ ; otherwise it is considered rejected. Let  $S$  denote the set of scheduled jobs and  $\bar{S}$  denote the set of rejected jobs. The goal is to minimize the sum of the weighted completion times of the scheduled jobs plus the total profits of the rejected jobs. Hence job profits can be equivalently interpreted as rejection penalties. We represent the problem using the scheduling notation introduced by Graham et al. [10] as:

$$1 \mid p_i = 1, \bar{d}_i = \bar{d} \mid \sum_S w_i C_i + \sum_{\bar{S}} e_i . \quad (1)$$

We assume that the number of jobs is at least  $\bar{d}$ . If not, letting  $U_+$  (resp.,  $U_-$ ) denoting the set of the jobs with nonnegative (resp., negative) weights, it is easy

---

This research was supported by NSF Grant CCF-1217980.



to observe that there exists a solution in which each job in  $U_+$  (resp.,  $U_-$ ) that is not rejected is scheduled in one of the first  $|U_+|$  (resp., last  $|U_-|$ ) slots. Using this observation, we can solve the given instance by solving two smaller instances for which our assumption is satisfied.

Like many other scheduling problems involving unit jobs, Problem 1 can be solved in polynomial time by a reduction to the maximum weight matching problem in bipartite graphs. Our contribution is an  $O(n \log n)$ -time algorithm for Problem 1 where  $n$  denotes the number of jobs. Engels et al. [6] give a pseudo-polynomial-time dynamic programming algorithm for the same objective except that variable processing times are allowed and no deadline restriction is imposed. Engels et al. first show that the decision version of the problem is NP-complete and then give an FPTAS. They also remark that a running time of  $O(n^2)$  can be achieved for the special case of unit processing times; our work improves this bound to  $O(n \log n)$ .

More general cases of Problem 1, almost all dealing with variable processing times, have been studied extensively. One of the earliest works that considers job specific profits and lateness penalties [20] reduces to our problem in the special case of setting all processing times to 1 and all due dates to 0. Two recent surveys review the research on various scheduling problems in which it is typically necessary to reject some of the jobs in order to achieve optimality [16, 19]. Epstein et al. [7] focus on unit jobs but consider only the online version of the problem. Shabtay et al. [17] split the scheduling objective into two criteria: the scheduling cost, which depends on the completion times of the jobs, and the rejection cost, which is the sum of the penalties paid for the rejected jobs. In addition to optimizing the sum of these two criteria, the authors study other variations of the problem such as optimizing one criterion while constraining the other, or identifying all Pareto-optimal solutions for the two criteria. The scheduling cost in that work is not exactly the weighted sum of the completion times; however, several other similar objectives are considered. We show that our problem becomes NP-hard if we split our criteria in the same manner and aim for optimizing one while bounding the other.

Given the improvement in running time that we achieve for Problem 1, it is natural to ask whether our approach can be adapted to obtain fast algorithms for interesting variants of Problem 1. We show the following generalization of Problem 1 can also be solved in  $O(n \log n)$  time:

$$1 \mid p_i = 1, d_i = d, \bar{d}_i = \bar{d} \mid \sum_S w_i C_i + c \sum_S w_i T_i + \sum_{\bar{S}} e_i . \tag{2}$$

In Problem 2, every job also has a common due date  $d$ , and completing a job after the due date incurs an additional tardiness penalty that depends on its weight and a positive constant  $c$ . The tardiness of a job is defined as  $T_i = \max\{0, C_i - d\}$ . Similar to Problem 1, we assume that the number of jobs is at least  $\bar{d}$ .

We solve Problems 1 and 2 by finding a maximum weight matching (MWM) in a complete bipartite graph that represents the scheduling instance. Due to the special structure of the edge weights, the space required to represent this

graph is linear in the number of vertices. Thus, aside from the scheduling applications, this work contributes to the research aimed at developing quasilinear algorithms for matching problems in compactly representable bipartite graphs. Both unweighted and vertex-weighted matching problems in convex bipartite graphs, the graphs in which the right vertices can be enumerated such that the neighbors of each left vertex are consecutive, have been studied extensively [8, 9, 12, 14, 21]. Plaxton [15] studies vertex-weighted matchings in two-directional orthogonal ray graphs, which generalize convex bipartite graphs. In contrast, the current paper focuses on a class of compactly representable bipartite graphs that is simpler in terms of the underlying graph structure (all edges are present), but allows for more complex edge weights.

The cost (distance) matrix of the complete bipartite graph that we construct for solving Problems 1 and 2 is a Monge matrix. An  $n \times m$  matrix  $C = (c_{ij})$  is called a Monge matrix if  $c_{ij} + c_{rs} \leq c_{is} + c_{rj}$  for  $1 \leq i < r \leq n$ ,  $1 \leq j < s \leq m$ . Burkard [2] provides a survey of the rich literature on applications of Monge structures in combinatorial optimization problems. When the cost matrix of a bipartite graph is a Monge matrix, an optimal maximum cardinality matching can be found in  $O(nm)$  time where  $n$  is the number of rows and  $m$  is the number of columns. If  $n = m$  then the diagonal of the cost matrix is a trivial solution. Aggarwal et al. [1] study several weighted bipartite matching problems where, aside being a Monge matrix, additional structural properties are assumed for the cost matrix. The authors present an  $O(n \log m)$ -time divide and conquer algorithm for the case where the number of rows  $n$  is at most the number of columns  $m$  and each row is bitonic, i.e., each row is a non-increasing sequence followed by a non-decreasing sequence. If we represent the edge weights of the bipartite graph that we construct for solving our problems in a matrix so that the rows correspond to the jobs and the columns correspond to the time slots, then both the Monge property and the bitonicity property are satisfied; in fact each row is monotonic. However, we end up having more rows than columns, which renders the algorithm of [1] inapplicable for our problems. If we had more columns than rows, as assumed in [1], then we would have a trivial solution which could be constructed by sorting the jobs with respect to their weights. In summary, similar to [1], our algorithm efficiently solves the weighted bipartite matching problem for Monge matrices having an additional structure on the rows. In contrast, the structural assumption we place on the rows is stronger than that of [1], and we require more rows than columns, whereas [1] requires the opposite.

Another application of bipartite graphs is in the context of unit-demand auctions. In a unit-demand auction, a collection of items is to be distributed among several bidders and each bidder is to receive at most one item [4, 13, 18]. Each bidder has a private value for each item, and submits to the auction a unit-demand bid that specifies a separate offer for each item. The VCG mechanism can be used to determine the outcome of a unit-demand auction, i.e., allocation and pricing of the items. The VCG allocation corresponds to an (arbitrary) MWM of the bipartite graph in which each left vertex represents a bid, each

right vertex represents an item, and the weight of the edge from a bid  $u$  to an item  $v$  represents the offer of the bid  $u$  for item  $v$ . The VCG mechanism is known to enjoy a number of desirable properties including efficiency, envy-freeness, and strategyproofness. Another contribution of this paper is an  $O(n \log n)$ -time algorithm for computing the VCG prices, given a VCG allocation of an auction instance that can be represented by a more general class of the complete bipartite graphs than the ones that we construct to solve Problems 1 and 2.

**Organization.** Section 2 describes the fast  $O(n \log n)$ -time algorithm for Problem 1. Section 3 describes how to extend the algorithm to solve Problem 2 within the same time bound. Section 4 views the problem from a unit-demand auction perspective and briefly presents the approach we take in the  $O(n \log n)$ -time algorithm for computing the VCG prices. Due to space limitations, some details are omitted from this conference version. The companion technical report [5] includes all of the material in the present version plus five appendices. Some of the proofs related to the algorithm for Problem 1 and a brief implementation are deferred to App. A [5]. The details of the extension for Problem 2 are explained in App. B [5]. Appendix C [5] presents the algorithm for computing the VCG prices in detail. Finally, App. E [5] proves the NP-hardness of the bicriteria variations of Problem 1 via reductions from the partition problem.

## 2 A Fast Algorithm for Problem 1

We encode an instance of Problem 1 as a weighted matching problem on a graph drawn from a certain family. Below we define this family, which we call  $\mathcal{G}$ , and we discuss how to express an instance of Problem 1 in terms of a graph in  $\mathcal{G}$ .

We define  $\mathcal{G}$  as the family of all complete edge-weighted bipartite graphs  $G = (U, V, w)$  such that the following conditions hold:  $|U| \geq |V|$ ; each left vertex  $u$  in  $U$  has two associated integers  $u.profit$  and  $u.priority$ ; the left vertices are indexed from 1 in non-decreasing order of priorities, breaking ties arbitrarily; right vertices are indexed from 1; the weight  $w(u, v)$  of the edge between a left vertex  $u$  and a right vertex  $v$  is equal to  $u.profit + u.priority \cdot j$  where  $j$  denotes the index of  $v$ . Note that a graph  $G = (U, V, w)$  in  $\mathcal{G}$  admits an  $O(|U|)$ -space representation.

Let  $I$  be an instance of Problem 1. The instance  $I$  consists of a set of  $n$  jobs to schedule, each with a profit and a weight, and a common deadline  $\bar{d}$  where we assume that  $n \geq \bar{d}$  as discussed in Sect. 1. We encode the instance  $I$  as a graph  $G = (U, V, w)$  in  $\mathcal{G}$  such that the following conditions hold:  $|U| = n$ ;  $|V| = \bar{d}$ ; each left vertex represents a distinct job in  $I$ ; each right vertex represents a time slot in which a job in  $I$  can be scheduled; for each job in  $I$  and the vertex  $u$  that represents that job,  $u.profit$  is equal to the profit of the job and  $u.priority$  is equal to the negated weight of the job. It is easy to see by inspecting the objective of Problem 1 that minimizing the weighted sum of completion times is equivalent to maximizing the same expression with negated weights, and minimizing the sum of the profits of the rejected jobs is equivalent to maximizing the sum of

the profits of the scheduled jobs. Hence, instance  $I$  of Problem 1 is equivalent to the problem of finding a maximum weight matching (MWM) of a graph  $G = (U, V, w)$  in  $\mathcal{G}$  that encodes  $I$ . Given this correspondence between the two problems, we refer to the left vertices (resp., right vertices) of a graph in  $\mathcal{G}$  as *jobs* (resp., *slots*). The problem of computing an MWM of a graph  $G = (U, V, w)$  in  $\mathcal{G}$  can be reduced to the maximum weight maximum cardinality matching (MWMCM) problem by adding  $|V|$  dummy jobs, each with profit and priority zero, to obtain a graph that also belongs to  $\mathcal{G}$ .

As a result of the equivalence of the two problems mentioned above and the reduction from the MWM to the MWMCM problem, we can obtain an  $O(n \log n)$ -time algorithm for Problem 1 by providing an  $O(|U| \log |U|)$ -time algorithm to compute an MWMCM of a graph  $G = (U, V, w)$  in  $\mathcal{G}$ . Before discussing this algorithm further, we introduce some useful definitions.

Let  $G = (U, V, w)$  be a graph in  $\mathcal{G}$ . We say that a subset  $U'$  of  $U$  is *optimal* for  $G$  if there exists an MWMCM  $M$  of  $G$  such that the set of jobs that are matched in  $M$  is equal to  $U'$ . Lemma 1 below shows that it is straightforward to efficiently construct an MWMCM of  $G$  given an optimal set of jobs for  $G$ . Let  $U'$  be a subset of  $U$  with size  $|V|$  and let  $i_1 < \dots < i_{|V|}$  denote the indices of the jobs in  $U'$ . Then we define  $\text{matching}(U')$  as the set of  $|V|$  job-slot pairs obtained by pairing the job with index  $i_k$  to the slot with index  $k$  for  $1 \leq k \leq |V|$ . The following lemma is a straightforward application of the rearrangement inequality [11, Section 10.2, Theorem 368] to our setting.

**Lemma 1.** *Let  $G = (U, V, w)$  be a graph in  $\mathcal{G}$ . Let  $U'$  be a subset of  $U$  with size  $|V|$ . Let  $W$  denote the maximum weight of any MCM of  $G$  that matches  $U'$ . Then  $\text{matching}(U')$  is of weight  $W$ .*

Having established Lemma 1, it remains to show how to efficiently identify an optimal set of jobs for a given graph  $G = (U, V, w)$  in  $\mathcal{G}$ . The main technical result of this section is an  $O(|U| \log |U|)$ -time dynamic programming algorithm for accomplishing this task. The following definitions are useful for describing our dynamic programming framework.

Let  $G = (U, V, w)$  be a graph in  $\mathcal{G}$ . For any integer  $i$  such that  $0 \leq i \leq |U|$ , we define  $U_i$  as the set of jobs with indices 1 through  $i$ . Similarly, for any integer  $j$  such that  $0 \leq j \leq |V|$ , we define  $V_j$  as the set of slots with indices 1 through  $j$ . For any integers  $i$  and  $j$  such that  $0 \leq j \leq i \leq |U|$  and  $j \leq |V|$ , we define  $G_{i,j}$  as the subgraph of  $G$  induced by the vertices  $U_i \cup V_j$ , and we define  $W(i, j)$  as the weight of an MWMCM of  $G_{i,j}$ . Note that any subgraph  $G_{i,j}$  of  $G$  also belongs to  $\mathcal{G}$ .

Let us define  $\mathcal{G}^*$  as the family of all graphs in  $\mathcal{G}$  having an equal number of slots and jobs. Given a graph  $G = (U, V, w)$  in  $\mathcal{G}^*$ , our dynamic programming algorithm computes in  $O(|U| \log |U|)$  total time an optimal set of jobs for each  $G_{|U|,j}$  for  $1 \leq j \leq |U|$ . For any graph  $G' = (U, V', w')$  in  $\mathcal{G}$ , we can construct a graph  $G = (U, V, w)$  in  $\mathcal{G}^*$  satisfying  $G'_{|U|,j} = G_{|U|,j}$  for all  $1 \leq j \leq |V'|$  by defining  $V$  as the set of  $|U|$  slots indexed from 1 through  $|U|$ . Thus, given any graph  $G' = (U, V', w')$  in  $\mathcal{G}$ , our algorithm can be used to identify an optimal set of jobs for each subgraph  $G'_{|U|,j}$  for  $1 \leq j \leq |V'|$  in  $O(|U| \log |U|)$  total time.

Throughout the remainder of this section, we fix a graph instance  $G = (U, V, w)$  in  $\mathcal{G}^*$ . The presentation of the algorithm is organized as follows. Section 2.1 introduces the core concept, which we call the *acceptance order*, that our algorithm is built on. Section 2.2 presents the key idea (Lemma 5) underlying our algorithm for computing the acceptance order. Finally, Sect. 2.3 describes an efficient augmented binary search tree implementation of the algorithm.

## 2.1 Acceptance Orders

Lemma 1 reduces Problem 1 to the problem of identifying an optimal subset of  $U$  for  $G$ . In addition to an optimal set of jobs for  $G$ , our algorithm determines for each integer  $i$  and  $j$  such that  $0 \leq j \leq i \leq |U|$ , a subset  $best(i, j)$  of  $U_i$  that is optimal for  $G_{i,j}$  (Lemma 3). There are quadratically many such sets, so in order to run in quasilinear time, we compute a compact representation of those sets by exploiting the following two properties. The first property is that  $best(i, j - 1)$  is a subset of  $best(i, j)$  for  $1 \leq j \leq i \leq |U|$ . Thus, for a fixed  $i$ , the sequence of sets  $best(i, 1), \dots, best(i, i)$  induces an ordering  $\sigma_i$  of jobs  $U_i$ , which we later define as the acceptance order of  $U_i$ , where the job at position  $j$  of  $\sigma_i$  is the one that is present in  $best(i, j)$  but not in  $best(i, j - 1)$ . The second property is that  $\sigma_{i-1}$  is a subsequence of  $\sigma_i$  for  $1 \leq i \leq |U|$ . This second property suggests an incremental computation of  $\sigma_i$ 's which will be exploited to find the weights of MWMCMs for all prefixes of jobs to solve Problem 2, as described in Sect. 3.

We now give the formal definitions of the acceptance order and the optimal set  $best(i, j)$ , and present two associated lemmas. The proofs of these two lemmas are provided in the companion technical report [5, Appendix A.1].

We say that a vertex is *essential* for an edge-weighted bipartite graph  $G$  if it belongs to every MWMCM of  $G$ .

For any integer  $i$  such that  $0 \leq i \leq |U|$  we define  $\sigma_i$  inductively as follows:  $\sigma_0$  is the empty sequence; for  $i > 0$  let  $u$  denote the job with index  $i$ , then  $\sigma_i$  is obtained from  $\sigma_{i-1}$  by inserting job  $u$  immediately after the prefix of  $\sigma_{i-1}$  of length  $p - 1$  where  $p$ , which we call the position of  $u$  in  $\sigma_i$ , is the minimum positive integer such that job  $u$  is essential for  $G_{i,p}$ . It is easy to see that  $\sigma_i$  is a sequence of length  $i$  and that  $1 \leq p \leq i$  since  $u$  is trivially essential for  $G_{i,i}$ . Furthermore,  $\sigma_{i-1}$  is a subsequence of  $\sigma_i$  for  $1 \leq i \leq |U|$ , as claimed above.

We say that  $\sigma_i$  is the *acceptance order* of the set of jobs  $U_i$ . Note that  $\sigma_{|U|}$  is the acceptance order of the set of all jobs.

**Lemma 2.** *Let  $i$  and  $j$  be any integers such that  $1 \leq j \leq i \leq |U|$  and let  $u$  denote the job with index  $i$ . Then job  $u$  is essential for  $G_{i,j}$  if and only if the position of  $u$  in  $\sigma_i$  is at most  $j$ .*

For any integers  $i$  and  $j$  such that  $0 \leq j \leq i \leq |U|$ , we define  $best(i, j)$  as the set of the first  $j$  jobs in  $\sigma_i$ . Thus,  $best(i, j - 1)$  is a subset of  $best(i, j)$  for  $1 \leq j \leq i \leq |U|$ , as claimed above.

**Lemma 3.** *Let  $i$  and  $j$  be any integers such that  $0 \leq j \leq i \leq |U|$ . Then  $matching(best(i, j))$  is an MWMCM of  $G_{i,j}$ .*

Lemmas 1 and 3 imply that once we compute the acceptance order  $\sigma_{|U|}$ , we can sort its first  $\bar{d}$  jobs by their indices to obtain a matching to solve Problem 1.

## 2.2 Computing the Acceptance Order

As we have established the importance of the acceptance order  $\sigma_{|U|}$ , we now describe how to compute it efficiently. We start with  $\sigma_1$  and introduce the tasks one by one in index order to compute the sequences  $\sigma_2, \dots, \sigma_{|U|}$  incrementally. Once we know  $\sigma_{i-1}$ , we just need to find out where to insert the job with index  $i$  in order to compute  $\sigma_i$ . We first introduce some definitions and a lemma, whose proof is provided in the companion technical report [5, Appendix A.1], and then we describe the key idea (Lemma 5) for finding the position of a job in the corresponding acceptance order.

For any integers  $i$  and  $j$  such that  $1 \leq j \leq i \leq |U|$ , let  $\sigma_i[j]$  denote the job with position  $j$  in  $\sigma_i$ , where  $\sigma_i[1]$  is the first job in  $\sigma_i$ .

For any job  $u$  that belongs to  $U$ , we define  $\text{better}(u)$  as the set of jobs that precede  $u$  in  $\sigma_i$  where  $i$  denotes the index of  $u$ . Thus  $|\text{better}(u)| = p - 1$  where  $p$  is the position of  $u$  in  $\sigma_i$ . The set  $\text{better}(u)$  is the set of jobs that precede  $u$  both in index order and in acceptance order.

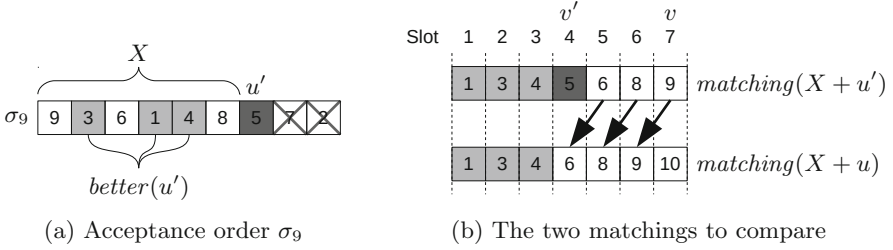
**Lemma 4.** *Let  $i$  and  $j$  be integers such that  $1 \leq j \leq i \leq |U|$ , and let  $i'$  denote the index of job  $\sigma_i[j]$ . Then the set of jobs in  $\text{best}(i, j - 1)$  with indices less than  $i'$  is equal to  $\text{better}(\sigma_i[j])$ .*

For any subset  $U'$  of  $U$ , we define  $\text{sum}(U')$  as  $\sum_{u \in U'} u.\text{priority}$ .

Now we are ready to discuss the idea behind the efficient computation of the acceptance orders incrementally. Assume that we already know the acceptance order  $\sigma_{i-1}$  of the set of the first  $i-1$  jobs for some integer  $i$  such that  $1 < i \leq |U|$ . Let  $u$  denote the job with index  $i$ . If we can determine in constant time, for any job in the set  $U_{i-1}$ , whether  $u$  precedes that job in  $\sigma_i$ , then we can perform a binary search in order to find in logarithmic time the position of  $u$  in  $\sigma_i$ . Suppose that we would like to know whether  $u$  precedes  $\sigma_{i-1}[j]$  in  $\sigma_i$  for some integer  $j$  such that  $1 \leq j < i$ . In other words we would like to determine whether the position of  $u$  in  $\sigma_i$  is at most  $j$ . In what follows, let  $u'$  denote the job  $\sigma_{i-1}[j]$  and let  $v$  denote the slot with index  $j$ . Then by Lemma 2, job  $u$  precedes  $u'$  in  $\sigma_i$  if and only if  $u$  is essential for  $G_{i,j}$ .

In order to determine whether job  $u$  is essential for  $G_{i,j}$ , we need to compare the weight of a heaviest possible matching for  $G_{i,j}$  that does not include  $u$  to the weight of a heaviest possible matching for  $G_{i,j}$  that includes  $u$ . The former weight is  $W(i-1, j)$ . Since job  $u$  has the highest index among the jobs with indices 1 through  $i$ , by Lemma 1, the latter weight is equal to  $w(u, v) + W(i-1, j-1)$ .

Let  $X$  denote  $\text{best}(i-1, j-1)$ . Since  $\text{best}(i-1, j-1) + u' = \text{best}(i-1, j)$ , Lemma 3 implies that the weight of  $\text{matching}(X + u')$  is equal to  $W(i-1, j)$ . By Lemma 3, the weight of  $\text{matching}(X)$  is  $W(i-1, j-1)$ . Since job  $u$  has the highest index among the jobs in  $X + u$ , the weight of  $\text{matching}(X + u)$  is  $w(u, v) + W(i-1, j-1)$ .



**Fig. 1.** An example in which we try to determine whether the job with index 10 precedes  $\sigma_9[7]$  in  $\sigma_{10}$ . Each box represents the job whose index is shown inside.

Combining the results of the preceding paragraphs, we conclude that job  $u$  is essential for  $G_{i,j}$  if and only if the weight of  $matching(X + u)$  is greater than the weight of  $matching(X + u')$ .

Figure 1 shows an example where  $i = 10$  and  $j = 7$ . Thus we are trying to determine whether the job with index 10 precedes  $\sigma_9[7]$  in  $\sigma_{10}$ . In this example,  $u$  denotes the job with index 10 and  $u'$  denotes  $\sigma_9[7]$ , which is the job with index 5, as shown in Fig. 1a. The set  $X$  is the first 6 jobs in  $\sigma_9$ . The jobs appearing past  $u'$  in  $\sigma_9$ , jobs with indices 7 and 2, do not participate in the matchings that we are interested in so they are crossed out. Figure 1b shows the two matchings  $matching(X + u')$  and  $matching(X + u)$  of which we would like to compare the weights. As seen in Fig. 1b, each job in  $X$  with index less than that of job  $u'$ , shaded light gray in the figure, is matched to the same slot in both  $matching(X + u)$  and  $matching(X + u')$ . By Lemma 4, those jobs are the ones in the set  $better(u')$ , which are the jobs with indices 1, 3 and 4 in the example. Hence job  $u'$  occurs in position  $|better(u')| + 1$  when we sort the set of jobs  $X + u'$  by index and thus it is matched to the slot with index  $|better(u')| + 1$  in  $matching(X + u')$ . Moreover, each job in  $X$  with index greater than that of job  $u'$  is matched to a slot with index one lower in  $matching(X + u)$  than in  $matching(X + u')$ , as depicted by the arrows in Fig. 1b for the jobs with indices 6, 8, and 9.

Hence the weight of  $matching(X + u)$  minus the weight of  $matching(X + u')$  is equal to  $w(u, v) - w(u', v')$  plus the sum of the priorities of all jobs in  $best(i - 1, j - 1)$  with indices greater than that of  $u'$ , where  $v'$  denotes the slot with index  $|better(u')| + 1$ . By Lemma 4, the latter sum is equal to  $sum(best(i - 1, j - 1)) - sum(better(u'))$ . These observations establish the proof of the following lemma which we utilize in computing the acceptance orders incrementally.

**Lemma 5.** *Let  $i$  and  $j$  be integers such that  $1 \leq j < i \leq |U|$ . Let  $u$  denote the job with index  $i$  and let  $u'$  denote the job  $\sigma_{i-1}[j]$ . Then the following are equivalent: (1) The position of  $u$  in  $\sigma_i$  is at most  $j$ ; (2) Job  $u$  is essential for  $G_{i,j}$ ; (3) The weight of  $matching(best(i - 1, j - 1) + u)$  is greater than the weight of  $matching(best(i - 1, j - 1) + u')$ ; and (4)  $w(u, v) > w(u', v') + sum(best(i - 1,$*

$j - 1)) - \text{sum}(\text{better}(u'))$  where  $v$  denotes the slot with index  $j$  and  $v'$  denotes the slot with index  $|\text{better}(u')| + 1$ .

### 2.3 Binary Search Tree Implementation

We obtain an efficient algorithm utilizing a self-balancing augmented binary search tree (BST) for incrementally computing the acceptance orders by a suitable choice of ordering the jobs, and an augmentation that is crucial in applying Lemma 5 in constant time. The jobs are stored in the BST so that an inorder traversal of the BST yields the acceptance order. The algorithm runs  $|U|$  iterations where the job with index  $i$  is inserted into the BST at iteration  $i$  to obtain  $\sigma_i$  from  $\sigma_{i-1}$  by performing a binary search. We first give some definitions that are useful in the description of the algorithm and then we state in Lemma 6 how to perform the comparisons for the binary search.

For a binary tree  $T$  and an integer  $i$  such that  $1 \leq i \leq |U|$ , we define the predicate  $\text{ordered}(T, i)$  to hold if  $T$  contains  $i$  nodes that represent the jobs  $U_i$ , and the sequence of the associated jobs resulting from an inorder traversal of  $T$  is  $\sigma_i$ . The job represented by a node  $x$  is denoted by  $x.\text{job}$ .

Let  $T$  be a binary tree satisfying  $\text{ordered}(T, i)$  for some  $i$ . For any node  $x$  in  $T$ ,  $\text{precede}(x, T)$  is defined as the set of jobs associated with the nodes that precede  $x$  in an inorder traversal of  $T$ .

**Lemma 6.** *Let  $i$  be an integer such that  $1 < i \leq |U|$  and let  $u$  denote the job with index  $i$ . Let  $T$  be a binary tree satisfying  $\text{ordered}(T, i - 1)$  and let  $x$  be a node in  $T$ . Assume that  $|\text{precede}(x, T)|$ ,  $\text{sum}(\text{precede}(x, T))$ ,  $|\text{better}(x.\text{job})|$ , and  $\text{sum}(\text{better}(x.\text{job}))$  are given. Then we can determine in constant time whether  $u$  precedes  $x.\text{job}$  in  $\sigma_i$ .*

*Proof.* Let  $j$  denote  $|\text{precede}(x, T)| + 1$ . Then  $\text{ordered}(T, i - 1)$  implies that  $x.\text{job}$  is  $\sigma_{i-1}[j]$  and  $\text{sum}(\text{precede}(x, T))$  is equal to  $\text{sum}(\text{best}(i - 1, j - 1))$ . Now let  $u'$  denote  $\sigma_{i-1}[j]$ . Then we can test Inequality 4 of Lemma 5 in constant time to determine whether the position of  $u$  in  $\sigma_i$  is at most  $j$ , thus whether  $u$  precedes  $u'$  in  $\sigma_i$ .  $\square$

Lemma 6 implies that once we know certain quantities about a node  $x$  in the BST then we can tell in constant time whether the new job precedes  $x.\text{job}$  in the acceptance order. The necessary information to compute the first two of those quantities can be maintained by standard BST augmentation techniques as described in [3, Chapter 14]. The other two quantities turn out to be equal to the first two at the time the node is inserted into the BST and they can be stored along with the node. The details are in the proof of the following result, which is presented together with a concise implementation in the companion technical report [5, Appendices A.2 and A.3].

**Theorem 1.** *The acceptance order of  $U$  can be computed in  $O(|U| \log |U|)$  time.*



As mentioned earlier, once  $\sigma_{|U|}$  is computed, we can extract an MWMCM of  $G_{|U|,j}$  for any  $j$  such that  $1 \leq j \leq |U|$ . If we are only interested in solutions for  $j$  up to some given  $m$ , then the algorithm can be implemented in  $O(n \log m)$  time by keeping at most  $m$  nodes in the BST. We achieve this by deleting the rightmost node when the number of nodes exceeds  $m$ . Note that if the jobs are not already sorted by priorities then we still need to spend  $O(n \log n)$  time.

If we would like to find out the weights of the MWMCMs of  $G_{|U|,j}$  for all  $j$  such that  $1 \leq j \leq |U|$ , a naive approach would be to sort all prefixes of  $\sigma_{|U|}$  and to compute the weights. The companion technical report [5, Appendix D] explains how to compute all those weights incrementally in linear time.

### 3 Introducing Tardiness Penalties

Given the improvement in running time that we achieve for Problem 1, we consider solving several variations of that problem and other related problems in more general families of compact bipartite graphs than the one we introduced in Sect. 2. A possible variation of Problem 1 is to allow a constant number of jobs to be scheduled in each time slot instead of only one. However, our approach of comparing the weights of two matchings that we illustrate in Fig. 1b fails because only some of the jobs, instead of all, in the set  $X$  having indices greater than the job we compare with are shifted to a lower slot. Solving this variation would enable us to address scheduling problems having symmetric earliness and tardiness penalties with respect to a common due date.

Another related problem is finding an MWM in a more general complete bipartite graph family that is still representable in space linear in the number of vertices. Consider the following extension to the complete bipartite graph  $G = (U, V, w)$  that is introduced in Sect. 2. For each slot (right vertex)  $v$  in  $V$ , we introduce an integer parameter  $v.quality$ . We assume that the slots are indexed from 1 in non-decreasing order of qualities, breaking ties arbitrarily. We allow an arbitrary number of slots that is less than the number of jobs. We also modify the edge weights so that  $w(u, v)$  between job  $u$  and slot  $v$  becomes  $u.profit + u.priority \cdot v.quality$ . While we have not been able to solve the MWM problem in such a graph faster than quadratic time yet, we describe in Sect. 4 how to compute the VCG prices given an MWM of such a graph that represents a unit-demand auction instance.

Here we describe a special case of the graph structure that is introduced in the previous paragraph. Suppose that the qualities of the slots form a non-decreasing sequence which is the concatenation of two arithmetic sequences. We are able to solve the MWM problem in such a graph instance, thus we solve Problem 2 introduced in Sect. 1 in  $O(n \log n)$  time. The key idea is to utilize the incremental computation of the acceptance orders so that we can find the weights of the MWMCMs between the slots whose qualities form the first arithmetic sequence (the slots before the common due date) and every possible prefix of jobs. Then we do the same between the slots whose qualities form the second arithmetic sequence (the slots after the common due date) and every possible suffix of jobs.

Then in linear time we find an optimal matching by determining which jobs to assign to the first group of slots and which jobs to the second group. The details are explained in the companion technical report [5, Appendix B].

## 4 Unit-Demand Auctions and VCG Prices

In this section, we view an instance  $G = (U, V, w)$  of the general complete bipartite graph family introduced in Sect. 3 from the perspective of unit-demand auctions. We refer to elements of  $U$  as bids and to elements of  $V$  as items. For any bid  $u$  and item  $v$ , the weight  $w(u, v)$  represents the amount offered by bid  $u$  to item  $v$ . We present an  $O(n \log n)$ -time algorithm for computing the VCG prices given a VCG allocation (an MWM of  $G$ ).

We review some standard definitions related to unit-demand auctions and we present the details of the algorithm in the companion technical report [5, Appendix C]. Here we briefly describe the approach we take in order to obtain the desired performance. One characterization of the VCG prices is that it is the minimum stable price vector [13]. Thus a naive algorithm would start with zero prices and then look for and eliminate the instabilities. While inspecting a particular instability, the algorithm would increase the prices just enough to eliminate that instability.

We take a similar approach that uses additional care. We start with a minimum price vector that does not cause an instability involving unassigned bids, by utilizing the geometric concept of the upper envelope. We then inspect the instabilities in a particular order, with two scans of the items, first in increasing and then in decreasing order of qualities. The most expensive step is the computation of the upper envelope, which takes  $O(n \log n)$  time.

**Acknowledgments.** In the early stages of this work we had developed an  $O(n \log^2 n)$ -time algorithm for the problem considered in Sect. 2. The authors wish to thank Eric Price for pointing out how to improve this bound to  $O(n \log n)$ , and for allowing us to include this improvement in the present paper.

## References

1. Aggarwal, A., Barnoy, A., Khuller, S., Kravets, D., Schieber, B.: Efficient minimum cost matching and transportation using the quadrangle inequality. *Journal of Algorithms* **19**(1), 116–143 (1995)
2. Burkard, R.E.: Monge properties, discrete convexity and applications. *European Journal of Operational Research* **176**(1), 1–14 (2007)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 3rd edn. The MIT Press (2009)
4. Demange, G., Gale, D., Sotomayor, M.A.O.: Multi-item auctions. *The Journal of Political Economy*, 863–872 (1986)
5. Domanic, N.O., Plaxton, C.G.: Scheduling unit jobs with a common deadline to minimize the sum of weighted completion times and rejection penalties. Tech. Rep. TR-14-11, Department of Computer Science, University of Texas at Austin (September 2014)

6. Engels, D.W., Karger, D.R., Kolliopoulos, S.G., Sengupta, S., Uma, R.N., Wein, J.: Techniques for scheduling with rejection. *Journal of Algorithms* **49**(1), 175–191 (2003)
7. Epstein, L., Noga, J., Woeginger, G.J.: On-line scheduling of unit time jobs with rejection: minimizing the total completion time. *Operations Research Letters* **30**(6), 415–420 (2002)
8. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences* **30**(2), 209–221 (1985)
9. Glover, F.: Maximum matching in a convex bipartite graph. *Naval Research Logistics Quarterly* **14**(3), 313–316 (1967)
10. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* **5**, 287–326 (1979)
11. Hardy, G.H., Littlewood, J.E., Pólya, G.: *Inequalities*, 2nd edn. Cambridge University Press (1952)
12. Katriel, I.: Matchings in node-weighted convex bipartite graphs. *INFORMS Journal on Computing* **20**, 205–211 (2008)
13. Leonard, H.B.: Elicitation of honest preferences for the assignment of individuals to positions. *The Journal of Political Economy*, 461–479 (1983)
14. Lipski Jr, W., Preparata, F.P.: Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica* **15**, 329–346 (1981)
15. Plaxton, C.G.: Vertex-weighted matching in two-directional orthogonal ray graphs. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) *Algorithms and Computation*. LNCS, vol. 8283, pp. 524–534. Springer, Heidelberg (2013)
16. Shabtay, D., Gaspar, N., Kaspi, M.: A survey on offline scheduling with rejection. *Journal of Scheduling* **16**(1), 3–28 (2013)
17. Shabtay, D., Gaspar, N., Yedidsion, L.: A bicriteria approach to scheduling a single machine with job rejection and positional penalties. *Journal of Combinatorial Optimization* **23**(4), 395–424 (2012)
18. Shapley, L.S., Shubik, M.: The assignment game I: The core. *International Journal of Game Theory* **1**(1), 111–130 (1971)
19. Slotnick, S.A.: Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research* **212**(1), 1–11 (2011)
20. Slotnick, S.A., Morton, T.E.: Selecting jobs for a heavily loaded shop with lateness penalties. *Computers and Operations Research* **23**(2), 131–140 (1996)
21. Steiner, G., Yeomans, J.S.: A linear time algorithm for determining maximum matchings in convex, bipartite graphs. *Computers and Mathematics with Applications* **31**, 91–96 (1996)

# **Combinatorial Optimization II**

# Solving Multi-choice Secretary Problem in Parallel: An Optimal Observation-Selection Protocol

Xiaoming Sun, Jia Zhang<sup>(✉)</sup>, and Jialin Zhang

Institute of Computing Technology, Chinese Academy of Sciences,  
Beijing, China  
zhangjia@ict.ac.cn

**Abstract.** The classical secretary problem investigates the question of how to hire the best secretary from  $n$  candidates who come in a uniformly random order. In this work we investigate a parallel generalizations of this problem introduced by Feldman and Tennenholtz [14]. We call it *shared  $Q$ -queue  $J$ -choice  $K$ -best secretary problem*. In this problem,  $n$  candidates are evenly distributed into  $Q$  queues, and instead of hiring the best one, the employer wants to hire  $J$  candidates among the best  $K$  persons. The  $J$  quotas are *shared* by all queues. This problem is a generalized version of  $J$ -choice  $K$ -best problem which has been extensively studied and it has more practical value as it characterizes the parallel situation.

Although a few of works have been done about this generalization, to the best of our knowledge, no optimal deterministic protocol was known with general  $Q$  queues. In this paper, we provide an optimal deterministic protocol for this problem. The protocol is in the same style of the  $\frac{1}{e}$ -solution for the classical secretary problem, but with multiple phases and adaptive criteria. Our protocol is very simple and efficient, and we show that several generalizations, such as the *fractional  $J$ -choice  $K$ -best secretary problem* and *exclusive  $Q$ -queue  $J$ -choice  $K$ -best secretary problem*, can be solved optimally by this protocol with slight modification and the latter one solves an open problem of Feldman and Tennenholtz [14]. In addition, we provide theoretical analysis for two typical cases, including the 1-queue 1-choice  $K$ -best problem and the shared 2-queue 2-choice 2-best problem. For the former, we prove a lower bound  $1 - O(\frac{\ln^2 K}{K^2})$  of the competitive ratio. For the latter, we show the optimal competitive ratio is  $\approx 0.372$  while previously the best known result is 0.356 [14].

## 1 Introduction

The classical *secretary problem* considers the situation that an employer wants to hire the best secretary from  $n$  candidates that come one by one in a uniformly random order [16]. Immediately after interviewing a candidate, the employer has to make an irrevocable decision of whether accepting this candidate or not. It

---

The work is partially supported by National Natural Science Foundation of China (61170062, 61222202, 61433014, 61173009).

is well known that the optimal solution is in a phase style: the employer firstly interviews  $n/e$  candidates without selecting anyone, then, he/she chooses the first candidate who is better than all previous ones. This protocol hires the best candidate with probability  $1/e$  and it is optimal [12, 29]. This problem captures many scenarios and has been studied extensively in many fields, such as decision theory [29], game theory [3, 20, 24] and theory of computation [6, 15], etc.

The classical secretary problem has many generalizations. Kleinberg [24] considered that employer selects multiple candidates with the objective to maximize the expectation of the total *values* of selected persons, and he proposed the first protocol whose expected competitive ratio tends to 1 when the number of choices goes to infinity. Buchbinder et al. [7] revealed an important relationship between the secretary problem and linear programming, which turns out to be a powerful method to construct optimal (randomized) protocols for many variants of secretary problems. Those variants include the so called  $J$ -choice  $K$ -best problem that the employer wants to hire  $J$  candidates from the best  $K$  candidates of all. Another important variant is proposed by Feldman et al. [14]. They were the first to introduce the parallel model. In their work, the candidates are divided into several queues to be interviewed by different interviewers. They studied two interesting settings: the quotas are pre-allocated and the quotas are shared by all interviewers. For these settings, they designed algorithms and analyzed the competitive ratios based on the random time arrival model [13]. Chan et al. [9] combined the results of Buchbinder et al. [7] with the random time arrival model [13] and considered infinite candidates. Under their model, they constructed a  $(J, K)$ -threshold algorithm for  $J$ -choice  $K$ -best problem. They also showed that their infinite model can be used to capture the asymptotic behavior of the finite model.

In this work, we focus on the shared parallel model introduced by Feldman et al. [14]. All the algorithms and analysis are based on the classical discrete and finite model. The parallel model can characterize many important situations where resource is limited or low latency is required. A typical case is the emergency diagnosis in hospital. To shorten the waiting time, patients are diagnosed by ordinary doctors in parallel. The serious patients are selected to be diagnosed by the expert doctors, since the experts are not enough and they can only deal with limited number of patients.

Our main result is an optimal deterministic protocol, which we call *Adaptive Observation-Selection Protocol*, for the *shared  $Q$ -queue  $J$ -choice  $K$ -best secretary problem* (abbreviated as *shared  $(Q, J, K)$  problem*). In this problem,  $n$  candidates are assigned to  $Q$  queues and interviewed in parallel. All queues *share* the  $J$  quotas. Besides, there is a set of weights  $\{w_k \mid 1 \leq k \leq K\}$  where  $w_k$  stands for how important the  $k$ -th rank is. The employer wants to maximize the expectation of the summation of the weight associated with the selected secretaries.

To design an optimal protocol, we generalize the linear program technique introduced by Buchbinder et al. [7]. Based on the optimal solution of LP model, one can design a randomized optimal algorithm. However, it is time consuming to solve the LP (the LP has  $nJK$  variables) and the randomized algorithm is unpractical to apply. Besides, although this LP model has been adopted in

many work, its structure hasn't been well studied in general. With digging into its structure, we develop a nearly linear time algorithm to solve the LP within  $O(nJK^2)$  time. More importantly, our protocol is deterministic. It is also simple and efficient. After solving the LP, the employer only spends  $O(\log K)$  time for each candidate. Our results answer the doubt in the work [7] that "the linear program which characterizes the performance may be too complex to obtain a simple mechanism". Our protocol can be extended to solve other extensions, as their LP models have the similar structure essentially. Among those extensions, the optimal protocol for *exclusive  $Q$ -queue  $J$ -choice  $K$ -best secretary problem* addresses an open problem in the work of Feldman et al. [14].

Our protocol is a nature extension of the well known  $1/e$ -protocol of the classical problem. In the  $1/e$ -protocol, the employer can treat the first  $n/e$  candidates as an *observation phase* and set the best candidate in this phase to be a criteria. In the second phase, the employer makes decision based on this criteria. In our problem, it is natural to extend the above idea to multiple phases in each queue and the criteria may change in different phases. Actually, the similar intuition has been used in many previous works, not only the secretary problem [2, 14], but also some other online problems such as online auction [20] and online matching [23]. This intuition seems straightforward, but it is hard to explain why it works. In this work, we theoretically prove that this intuition indicates the right way and can lead to optimality in our case.

Another contribution is that we provide theoretical analysis for the competitive ratio of non-weighted cases of our problem. For the  $(1, 1, K)$  case, we provide a lower bound  $1 - O\left(\frac{\ln^2 K}{K^2}\right)$  and some numerical results. For the *shared*  $(2, 2, 2)$  case, we show that the optimal competitive ratio is approximately 0.372 which is better than 0.356 that obtained by Feldman et al. [14].

Due to the space limitation, all the proofs in this work are omitted. Details can be see in the full version [31].

**More Related Work.** Ajtai et al. [1] have considered the  $K$ -best problem with the goal to minimize the expectation of the sum of the ranks (or powers of ranks) of the accepted objects. In the *Matroid secretary problem* [4, 8, 10, 11, 18, 21, 22, 25, 30], it introduces some combinatorial restrictions (called matroid restriction) to limit the possible set of selected secretaries. Another kind of combinatorial restriction is the knapsack constraints [2, 3]. They combined the online knapsack problem and the idea of random order in secretary problem. Another branch of works consider the value of selected secretaries. It is no longer the summation of values of each selected one, but will be a submodular function among them [5, 13, 19]. Besides, Feldman et al. [13] considered the secretary problem from another interesting view. They assumed all of the candidates come to the interview at a random time instead of a random order. Some works talked about the case that only partial order between candidates are known for the employer [17, 27]. There are also some works considering the secretary problem from the view of online auction [2–4, 20, 23, 24, 26, 28]. In these works, one seller wants to sell one of more identical items to  $n$  buyers, and the buyers will come to the market at different

time and may leave after sometime. The goal of the seller is to maximize his/her expected revenue as well as the concern of truthfulness.

## 2 Preliminaries

In this section we formally define the shared  $(Q, J, K)$  problem. Given positive integers  $Q, J, K$  and  $n$  with  $Q, J, K \leq n$ , suppose the employers want to hire  $J$  secretaries from  $n$  candidates that come one by one in a uniformly random order. There are  $Q$  interviewers. Due to practical reason, like time limitation, they do the interview in parallel. All candidates are divided into  $Q$  queues, that is, the  $i$ -th person is assigned to the queue numbered  $i \bmod Q$  ( $i = 1, \dots, n$ ). The employers then interview those candidates simultaneously. All the  $J$  quotas are shared by the  $Q$  queues. That means in each queue, the employers can hire a candidate if the total number of hired persons is less than  $J$ . The only information shared among  $Q$  queues is the number of the candidates already hired. Thus the employer in each queue only knows the relative order about those candidates already interviewed in his/her own queue but has no idea about those unseen ones and persons in other queues. After interviewing each candidate, the employer should make an irrevocable decision about whether employ this candidate or not. For the sake of fairness, we make a reasonable assumption that the duration of the interviewing for each candidate is uniform and fixed. This ensures the interview in each queue is carried out in the same pace. When employers in several queues want to hire the candidate in their own queues at the same time, to break the tie, the queues with smaller number have higher priority. Besides, we suppose the employers only value the best  $K$  candidates and assign different weights to every one of the  $K$  candidates and those weights satisfies  $w_1 \geq w_2 \geq \dots \geq w_K > 0$  where the  $w_k$  stands for the importance of the  $k$ -th best candidate in the employer's view. Candidates not in best  $K$  can be considered have a weight 0. The object function is to maximize the expectation of the summation of the weight of selected candidates. This is the so called *shared  $Q$ -queue  $J$ -choice  $K$ -best secretary problem*, and we abbreviate it as *shared  $(Q, J, K)$  problem* for convenience.

## 3 Optimal Protocol for Shared $(Q, J, K)$ Problem

In this section, we first characterize the shared  $(Q, J, K)$  problem by a linear program and then construct a deterministic protocol for the shared  $(Q, J, K)$  problem. We will talk about the relationship between the linear program and our protocol, and finally use the idea of primal and dual to show our protocol is optimal.

### 3.1 Linear Program for the Shared $(Q, J, K)$ Secretary Problem

We use a linear program to characterize the shared  $(Q, J, K)$  problem and provide its dual program. This approach was introduced by Buchbinder et al. [7] to model the  $J$ -choice  $K$ -best problem. We are the first to generalize it to the shared  $(Q, J, K)$  problem.



**Primal Program for the Shared  $(Q, J, K)$  Problem.** Without loss of generality, we assume  $n$  is a multiple of  $Q$ . Let  $c_{q,i}$  stand for the  $i$ -th candidate in  $q$ -th queue and  $x_{q,i}^{j|k}$  stand for the probability that  $c_{q,i}$  is selected as the  $j$ -th one given that he/she is the  $k$ -th best person up to now in  $q$ -th queue. When the  $J, K$  and the weights are given, we know the offline optimal solution is  $\sum_{l=1}^{\min(J,K)} w_l$ . We denote it as  $W$ . Then we can model the shared  $(Q, J, K)$  problem as follow.

$$\begin{aligned} \max z &= \frac{1}{nW} \sum_{q=1}^Q \sum_{j=1}^J \sum_{l=1}^K \sum_{i=1}^n \sum_{k=l}^K w_k \frac{\binom{i-1}{l-1} \binom{n-i}{k-i}}{\binom{n-1}{k-1}} x_{q,i}^{j|k} \\ \text{s.t.} \quad &\begin{cases} x_{q,i}^{j|k} \leq \sum_{m=1}^Q \sum_{s=1}^{i-1} \frac{1}{s} \sum_{l=1}^K \left( x_{m,s}^{j-1|l} - x_{m,s}^{j|l} \right) + \sum_{m=1}^{q-1} \frac{1}{i} \sum_{l=1}^K \left( x_{m,i}^{j-1|l} - x_{m,i}^{j|l} \right), \\ (1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq k \leq K, 1 \leq j \leq J) \\ x_{q,i}^{j|k} \geq 0, \quad (1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq k \leq K, 1 \leq j \leq J). \end{cases} \end{aligned} \tag{1}$$

For further analysis, we provide several definitions about the primal program.

**Definition 1 (Crucial Constraint).** We call the constraint

$$x_{q,i}^{j|k} \leq \sum_{m=1}^Q \sum_{s=1}^{i-1} \frac{1}{s} \sum_{l=1}^K \left( x_{m,s}^{j-1|l} - x_{m,s}^{j|l} \right) + \sum_{m=1}^{q-1} \frac{1}{i} \sum_{l=1}^K \left( x_{m,i}^{j-1|l} - x_{m,i}^{j|l} \right)$$

for  $1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq j \leq J, 1 \leq k \leq K$ , the crucial constraint for  $x_{q,i}^{j|k}$ .

**Definition 2 ((0, 1)-solution and Crucial Position).** Given a feasible solution of the primal program, if there are  $JKQ$  points  $\{i'_{q,j,k} \mid 1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K\}$  satisfy

$$x_{q,i}^{j|k} = \begin{cases} \sum_{m=1}^Q \sum_{s=1}^{i-1} \frac{1}{s} \sum_{l=1}^K \left( x_{m,s}^{j-1|l} - x_{m,s}^{j|l} \right) + \sum_{m=1}^{q-1} \frac{1}{i} \sum_{l=1}^K \left( x_{m,i}^{j-1|l} - x_{m,i}^{j|l} \right) > 0, & i'_{q,j,k} \leq i \leq n/Q \\ 0, & 1 \leq i < i'_{q,j,k} \end{cases}$$

for all  $1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K$ , we call this feasible solution (0, 1)-solution of the primal program, and  $i'_{q,j,k}$  is the crucial position for  $x_{q,i}^{j|k}$ .

Note that, in a (0, 1)-solution, only when  $x_{q,i}^{j|k} > 0$ , we consider the crucial constraint for the  $x_{q,i}^{j|k}$  is tight, otherwise, the crucial constraint is slack, even though the constraint may be actually tight when  $x_{q,i}^{j|k} = \sum_{m=1}^Q \sum_{s=1}^{i-1} \frac{1}{s} \sum_{l=1}^K \left( x_{m,s}^{j-1|l} - x_{m,s}^{j|l} \right) + \sum_{m=1}^{q-1} \frac{1}{i} \sum_{l=1}^K \left( x_{m,i}^{j-1|l} - x_{m,i}^{j|l} \right) = 0$ .

**Dual Program.** Suppose  $b_i^k = \sum_{l=k}^K w_l \frac{\binom{i-1}{k-1} \binom{n-i}{l-k}}{\binom{n-1}{l-1}}$ . We have the dual program:

$$\begin{aligned} \min z &= \sum_{q=1}^Q \sum_{k=1}^K \sum_{i=1}^{n/Q} y_{q,i}^{1|k} \\ \text{s.t.} \quad &\begin{cases} y_{q,i}^{j|k} + \frac{1}{i} \sum_{m=1}^Q \sum_{s=i+1}^{n/Q} \sum_{l=1}^K (y_{m,s}^{j|l} - y_{m,s}^{j+1|l}) + \frac{1}{i} \sum_{m=q+1}^Q \sum_{l=1}^K (y_{m,i}^{j|l} - y_{m,i}^{j+1|l}) \geq \frac{b_i^k}{nW}, \\ (1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq j \leq J, 1 \leq k \leq K) \\ y_{q,i}^{j|k} \geq 0, (1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq j \leq J, 1 \leq k \leq K). \end{cases} \end{aligned}$$

In this program, we add a set of dummy variables  $y_{q,i}^{(J+1)|k}$  and set them to be 0 for the brief. Respectively, we can define the *crucial constraint* and *crucial position* for the  $y_{q,i}^{j|k}$  and the (0, 1)-solution for this dual program.

### 3.2 Protocol Description

---

**Algorithm 1. Preprocessing Part**

---

```

input :  $n, J, K, Q, \{w_k \mid 1 \leq k \leq K\}$ 
output:  $\{i_{q,j,k} \mid 1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K\}$ 
1  $i_{q,j,k}$  ( $1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K$ ):  $JKQ$  crucial positions, initially 1
2  $y_{q,i}^{j|k}$  ( $1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq j \leq J + 1, 1 \leq k \leq K$ ): initially 0
3 for  $i = n/Q$  to 1 do
4   for  $q = Q$  to 1 do
5     for  $j = J$  to 1 do
6       for  $k = K$  to 1 do
7          $y_{q,i}^{j|k} \leftarrow \frac{b_i^k}{nW} + \frac{1}{i} \sum_{m=1}^Q \sum_{s=i+1}^{n/Q} \sum_{l=1}^K (y_{m,s}^{j+1|l} - y_{m,s}^{j|l}) +$ 
            $\frac{1}{i} \sum_{m=q+1}^Q \sum_{l=1}^K (y_{m,i}^{j+1|l} - y_{m,i}^{j|l})$ 
8         if  $y_{q,i}^{j|k} \leq 0$  then
9            $y_{q,i}^{j|k} \leftarrow 0$ 
10          if  $i = n$  or  $y_{q,i+1}^{j|k} > 0$  then
11             $i_{q,j,k} \leftarrow i + 1$  ▷ Find and record the crucial position

```

---

The protocol consists of two parts. The first part (Algorithm 1) takes  $J, K, Q$  and  $n$  as inputs and outputs  $JKQ$  positions  $\{i_{q,j,k} \mid 1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K\}$ . We will show some properties about these positions later.

The preprocessing part actually solves the dual program as defined in Section 3.1. But it is more efficient than the ordinary LP solver. It is easy to check if we calculate the value of  $y_{q,i}^{j|k}$  in line 7 carefully, the time complexity of the algorithm is  $O(nJK^2)$ .

The second part (Algorithm 2) takes the output of preprocessing part as input and does the interview on  $Q$  queues simultaneously. For each queue, this protocol consist of  $J$  rounds. When  $j$  ( $1 \leq j \leq J - 1$ ) persons were selected from all queues, the protocol will enter the  $(j + 1)$ -th round immediately. In each round, the protocol divided candidates in each queue into  $K + 1$  phases. For each queue, in the  $k$ -th ( $1 \leq k \leq K$ ) phase, that's from  $(i_{q,j,k-1})$ -th candidate to  $(i_{q,j,k} - 1)$ -th candidate, the protocol selects the  $(k - 1)$ -th best person of previous  $k - 1$  phases in this queue as criteria, and just hires the first one that better than this criteria. Candidates in each queue come up one by one. For each candidate, the employers check the number of candidates selected to determine the current round, and then query the current phase based on the position of current candidate, and finally make decision by comparing with criteria of this phase. The protocol will terminate when all candidates were interviewed or  $J$  candidates are selected. In the protocol, we define a *global* order which is consistent with the problem definition. Using  $c_{q,i}$  to stand for the  $i$ -th candidate of  $q$ -th queue. We say  $c_{q',i'}$  comes *before*  $c_{q,i}$  if  $i' < i$  or  $i' = i$  and  $q' < q$ .

---

**Algorithm 2. Adaptive Observation-Selection Protocol**

---

**input** :  $n, Q, J, K, \{i_{q,j,k} \mid 1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K\}$   
**output**: the selected persons

```

1 let  $i_{q,j,K+1}$  to be  $n + 1$ 
2 for all queues simultaneously do
3   ▷ suppose  $q$  is the number of an arbitrary queue
4   for  $i = 1$  to  $i_{q,1,1} - 1$  do interview without selecting anyone
5   for  $i = i_{q,1,1}$  to  $n/Q$  do
6     interview current candidate  $c_{q,i}$ 
7     let  $j$  to be the number of selected persons before  $c_{q,i}$  in global order
8     if  $j = J$  then return
9     let  $k$  to be the current phase number of  $(j + 1)$ -th round ▷ that's
10    the  $k$  satisfies  $i_{q,j+1,k-1} \leq i < i_{q,j+1,k}$ 
11    let  $s$  to be the  $(k - 1)$ -th best one from the first candidate to
12     $(i_{q,j+1,k-1})$ -th candidate
13    if  $c_{q,i}$  is better than  $s$  then
14      select  $c_{q,i}$ 

```

---

**3.3 Optimality of the Adaptive Observation-Selection Protocol**

In the rest of this work, we use  $y_{q,i}^{j|k*}$  to stand for the value of  $y_{q,i}^{j|k}$  obtained from the preprocessing part for  $1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq j \leq J + 1, 1 \leq k \leq K$ .

These two notations  $y_{q,i}^{j|k}$  and  $y_{q,i}^{j|k*}$  should be clearly distinguished. The former is a variable in the dual program, while the latter is a value we get from the preprocessing part.

**Preparations.** For the clarity of the proof, we distill some fundamental results in this part. The Proposition 1 talks about two properties of  $b_i^k$  defined in the dual program, and the Lemma 1, 2 reveal some important properties of the preprocessing part. The Lemma 3 considers a recurrence pattern. This recurrence can be used to explore the structure of the constraints of the dual program.

**Proposition 1.** For  $1 \leq k \leq K, 1 \leq i \leq n/Q, b_i^k$  satisfies (a)  $ib_i^k \leq (i + 1)b_{i+1}^k$  and (b)  $b_i^k \geq b_i^{k+1}$ .

**Lemma 1.** The  $\{i_{q,j,k} \mid 1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K\}$  obtained from the preprocessing part satisfies  $i_{q,j,t} \leq i_{q,j,k}$ , and we have  $y_{q,i}^{j|t*} \geq y_{q,i}^{j|k*} > 0$  for  $1 \leq t < k$ .

**Lemma 2.** According to the preprocessing part, if  $y_{q,i}^{j|k*} > 0$  and  $y_{q,i}^{j|k*} \geq y_{q,i}^{j+1|k*}$ , we have  $y_{q,i}^{j|t*} \geq y_{q,i}^{j+1|t*}$  for  $1 \leq t < k$ .

**Lemma 3.** Suppose  $m, t, Q, K$  are positive integers and  $c$  is a constant real number.  $\{f_t\}_{t=1}^m, \{g_t\}_{t=1}^m$  and  $\{h_t\}_{t=1}^m$  are three sequences. Let  $i = \lfloor \frac{t-1}{Q} \rfloor + 1$ , if the recursion  $f_t + \frac{K}{i} \sum_{s=t+1}^m (f_s - g_s) + \frac{c}{i} = h_i$  is held, then all the values in  $\{f_t\}_{t=1}^n$  will increase when  $c$  decreases or values in  $\{g_t\}_{t=1}^n$  increase.

**Main Frame of the Proof.** The main idea of the proof is described as follow. Firstly we show the fact that the Adaptive Observation-Selection protocol can be mapped to a feasible  $(0, 1)$ -solution of the primal program (Lemma 4) while the  $\{y_{q,i}^{j|k*} \mid 1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq j \leq J, 1 \leq k \leq K\}$  obtained from the preprocessing part is corresponding to a feasible  $(0, 1)$ -solution of the dual program (Lemma 5). Then, we argue that these two feasible  $(0, 1)$ -solutions satisfy the *theorem of complementary slackness* (Theorem 1). Thus both the solutions are optimal respectively. This means our protocol is optimal.

**Lemma 4.** Taking the  $\{i_{q,j,k} \mid 1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K\}$  obtained from the preprocessing part as input, the Adaptive Observation-Selection Protocol can be mapped to a  $(0, 1)$ -solution of the primal program and the  $i_{q,j,k}$  is the crucial position of  $x_{q,i}^{j|k}$ .

The proof is achieved by calculating the probability that  $c_{q,i}$  is selected in  $j$ -th round by the protocol given that he/she is the  $k$ -th best up to now, that's the  $x_{q,i}^{j|k}$ . We can find that the  $x_{q,i}^{j|k}$  obtained from the protocol exactly satisfies the definition of  $(0, 1)$ -solution.

The relationship between the preprocessing part and the dual program is the essential and most complicate part in this work. As the dual program is extremely complex, insight on the structure should be raised. The proof relies heavily on the properties of the preprocessing part and the dual program revealed in preparation part.

**Lemma 5.** *The  $\{y_{q,i}^{j|k*} \mid 1 \leq q \leq Q, 1 \leq i \leq n/Q, 1 \leq j \leq J, 1 \leq k \leq K\}$  obtained from the preprocessing part is a  $(0, 1)$ -solution of the dual program.*

The crucial positions play a key role in the protocol, and up to now, some properties of them have been revealed. We summarize those properties here.

**Proposition 2.** *For  $1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K$ , we have  $i_{q+1,j,k} \leq i_{q,j,k}$ ,  $i_{q,j,k} \leq i_{q,j+1,k}$ , and  $i_{q,j,k} \leq i_{q,j,k+1}$ .*

Employing the complementary slackness theorem, we can show the our protocol is optimal.

**Theorem 1.** *Taking the  $\{i_{q,j,k} \mid 1 \leq q \leq Q, 1 \leq j \leq J, 1 \leq k \leq K\}$  obtained from the preprocessing part as input, the Adaptive Observation-Selection Protocol is optimal for the shared  $(Q, J, K)$  problem.*

## 4 Extensions and Analysis of the Optimal Protocol

### 4.1 Applications in Other Generalizations

Our optimal protocol is based on the essential structure of the LP model. Several variants can be characterized by LP model with similar structure. Thus our optimal protocol can be extended to solve these related variants.

It is obvious that we can obtain an optimal protocol for weighted  $J$ -choice  $K$ -best secretary problem when  $Q$  is set to be 1. Based on the  $J$ -choice  $K$ -best problem, we consider another variant: the employer just interviews the first  $m$  candidates,  $1 \leq m \leq n$ , due to time or resource limitation. Other settings keep unchanged. We call this problem *fractional  $J$ -choice  $K$ -best secretary problem*. We can characterize this problem with LP program called *FLP* (see the full version). The *FLP* has the same structure with the LP 1, and all the properties used to show the optimality of the Adaptive Observation-Selection protocol are still held. Thus, our protocol can be easily generalized to solve this problem.

In the shared  $(Q, J, K)$  problem, all interviewers *share* the  $J$  quotas. Another case is that a fixed quota is preallocated to each queue, that's to say, in any queue  $q$ , the employer can only hire at most  $J_q$  candidates where  $J = \sum_{q=1}^Q J_q$ . Besides, we suppose there are  $n_q$  candidates in queue  $q$  so that  $n = \sum_{q=1}^Q n_q$ . Other settings, except the synchronous requirement, keep unchanged compared to the shared  $(Q, J, K)$  problem. This is the problem which is called *exclusive  $Q$ -queue  $J$ -choice  $K$ -best secretary problem* (abbreviated as *exclusive  $(Q, J, K)$  problem*). Feldman et al. [14] have considered the non-weighted version of the exclusive  $(Q, J, K)$  problem with the condition  $J = K$ . Actually, for each queue of the exclusive  $(Q, J, K)$  problem, since what we care about is the expectation and the candidates' information and quotas can not be shared, how employer selects candidate has *no influence* on other queues. So, it is an independent fractional weighted  $J_q$ -choice  $K$ -best secretary problem with  $m = n_q$  in each queue. Then, running the modified Adaptive Observation-Selection protocol on each queue is an optimal protocol for exclusive  $(Q, J, K)$  problem.

### 4.2 Competitive Ratio Analysis

Let  $\alpha(Q, J, K)$  stand for the competitive ratio of Adaptive Observation-Selection Protocol. For the general case,  $\alpha(Q, J, K)$  is complicated to analyze either from the view of protocol or the dual program. In this section, we provide analysis about two typical cases: the  $(1, 1, K)$  case and the  $(2, 2, 2)$  case. Both the cases we deal with are the uniformly weighted (or non-weighted) versions of shared  $(Q, J, K)$  problem, i.e.  $w_1 = w_2 = \dots = w_K = 1$ .

The first one we study is the  $(1, 1, K)$  case that selecting 1 candidate among the top  $K$  of  $n$  candidates with just one queue. It is also called  $K$ -best problem. For this case, we provide a simple 3-phase algorithm. This algorithm actually divides the candidates into 3 phases. In the first phase, the algorithm hires no one, and then in the second phases, it hires a candidate if he/she is the best one up to now. In the last phase, it hires a candidate if he/she is the best or the second best candidate so far. As our Adaptive Observation-Selection protocol is optimal, the performance of this 3-phase algorithm is a lower bound of our protocol. We get the following lower bound of  $\alpha(1, 1, K)$  based on the analysis of this three-phase algorithm.

**Theorem 2.**  $\alpha(1, 1, K) \geq 1 - O\left(\frac{\ln^2 K}{K^2}\right)$  when  $K$  is large enough and  $n \gg K$ .

The Adaptive Observation-Selection protocol performs much better in fact. Table 1 is the result of numerical experiment for small  $K$ . As we can see,  $\alpha(1, 1, K)$  goes to 1 sharply. But it is too complex to analyze when there are  $K + 1$  phases.

**Table 1.** The value of  $\alpha(1, 1, K)$  when  $n = 10000$

$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 6$	$K = 7$	$K = 8$	$K = 9$	$K = 10$
0.3679	0.5736	0.7083	0.7988	0.8604	0.9028	0.9321	0.9525	0.9667	0.9766
$K = 11$	$K = 12$	$K = 13$	$K = 14$	$K = 15$	$K = 16$	$K = 17$	$K = 18$	$K = 19$	$K = 20$
0.9835	0.9884	0.9918	0.9942	0.9959	0.9971	0.9980	0.9986	0.9990	0.9993
$K = 21$	$K = 22$	$K = 23$	$K = 24$	$K = 25$	$K = 26$	$K = 27$	$K = 28$	$K = 29$	$K = 30$
0.9995	0.9996	0.9997	0.9998	0.9999	0.9999	0.9999	>0.9999	>0.9999	>0.9999

Another case is when  $Q = J = K = 2$  and we have the following result.

**Theorem 3.** When  $n$  is large enough, the Adaptive Observation-Selection protocol achieves a competitive ratio  $\alpha(2, 2, 2) \approx 0.372$ .

The main idea is to calculate the optimal  $(0, 1)$ -solution of the dual program based on the preprocessing part. This analysis is almost accurate when  $n$  is large enough.

## 5 Conclusion

In this paper, we deal with a generalization of secretary problem in the parallel setting, the shared  $Q$ -queue  $J$ -choice  $K$ -best secretary problem, and provide a deterministic optimal protocol. This protocol can be applied to a series of relevant variants while keeps optimal. In addition, we provide some analytical results for two typical cases: the 1-queue 1-choice  $K$ -best case and the shared 2-queue 2-choice 2-best case.

There are several interesting open problems. The first one is making a tighter analysis of the competitive ratio for shared  $Q$ -queue  $J$ -choice  $K$ -best secretary problem. For the 1-queue 1-choice  $K$ -best case, we conjecture that the competitive ratio has the form of  $1 - O(f(K)^K)$  for some negligible function  $f$ . For the general case, there is no notable result up to now and lots of work remain to be done. Another interesting aspect is to know whether the technique in this paper can be used to find deterministic protocol for other variations such as matroid secretary problem, submodular secretary problem, knapsack secretary problem etc.

## References

1. Ajtai, M., Megiddo, N., Waarts, O.: Improved algorithms and analysis for secretary problems and generalizations. *SIAM Journal on Discrete Mathematics* **14**(1), 1–27 (2001)
2. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.D.: A knapsack secretary problem with applications. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) *RANDOM 2007 and APPROX 2007*. LNCS, vol. 4627, pp. 16–28. Springer, Heidelberg (2007)
3. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: Online auctions and generalized secretary problems. *SIGecom Exchanges* **7**(2), 7:1–7:11 (2008)
4. Babaioff, M., Immorlica, N., Kleinberg, R.: Matroids, secretary problems, and online mechanisms. In: *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 434–443 (2007)
5. Bateni, M., Hajiaghayi, M., Zadimoghaddam, M.: Submodular secretary problem and extensions. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) *APPROX 2010*. LNCS, vol. 6302, pp. 39–52. Springer, Heidelberg (2010)
6. Borosan, P., Shabbir, M.: A survey of secretary problem and its extensions (2009)
7. Buchbinder, N., Jain, K., Singh, M.: Secretary problems via linear programming. In: Eisenbrand, F., Shepherd, F.B. (eds.) *IPCO 2010*. LNCS, vol. 6080, pp. 163–176. Springer, Heidelberg (2010)
8. Chakraborty, S., Lachish, O.: Improved competitive ratio for the matroid secretary problem. In: *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1702–1712 (2012)
9. Hubert Chan, T.-H., Chen, F.: A primal-dual continuous lp method on the multi-choice multi-best secretary problem. *CoRR*, abs/1307.0624 (2013)
10. Dimitrov, N.B., Plaxton, C.G.: Competitive weighted matching in transversal matroids. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I*. LNCS, vol. 5125, pp. 397–408. Springer, Heidelberg (2008)

11. Dinitz, M., Kortsarz, G.: Matroid secretary for regular and decomposable matroids. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 108–117 (2013)
12. Dynkin, E.B.: The optimum choice of the instant for stopping a markov process. *Soviet Mathematics Doklady* **4** (1963)
13. Feldman, M., Naor, J.S., Schwartz, R.: Improved competitive ratios for submodular secretary problems (extended abstract). In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) RANDOM 2011 and APPROX 2011. LNCS, vol. 6845, pp. 218–229. Springer, Heidelberg (2011)
14. Feldman, M., Tennenholtz, M.: Interviewing secretaries in parallel. In: Proceedings of the 13th ACM Conference on Electronic Commerce, pp. 550–567 (2012)
15. Freeman, P.R.: The secretary problem and its extensions: A review. *International Statistical Review*, 189–206 (1983)
16. Gardner, M.: Mathematical games. *Scientific American*, 150–153 (1960)
17. Georgiou, N., Kuchta, M., Morayne, M., Niemiec, J.: On a universal best choice algorithm for partially ordered sets. *Random Structures and Algorithms* **32**(3), 263–273 (2008)
18. Gharan, S.O., Vondrák, J.: On variants of the matroid secretary problem. *Algorithmica* **67**(4), 472–497 (2013)
19. Gupta, A., Roth, A., Schoenebeck, G., Talwar, K.: Constrained non-monotone submodular maximization: offline and secretary algorithms. In: Saberi, A. (ed.) WINE 2010. LNCS, vol. 6484, pp. 246–257. Springer, Heidelberg (2010)
20. Hajiaghayi, M., Kleinberg, R., Parkes, D.C.: Adaptive limited-supply online auctions. In: Proceedings of the 5th ACM Conference on Electronic Commerce, pp. 71–80 (2004)
21. Im, S., Wang, Y.: Secretary problems: laminar matroid and interval scheduling. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1265–1274 (2011)
22. Jaillet, P., Soto, J.A., Zenklusen, R.: Advances on matroid secretary problems: free order model and laminar case. In: Goemans, M., Correa, J. (eds.) IPCO 2013. LNCS, vol. 7801, pp. 254–265. Springer, Heidelberg (2013)
23. Kesselheim, T., Radke, K., Tönnis, A., Vöcking, B.: An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 589–600. Springer, Heidelberg (2013)
24. Kleinberg, R.: A multiple-choice secretary algorithm with applications to online auctions. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 630–631 (2005)
25. Korula, N., Pál, M.: Algorithms for secretary problems on graphs and hypergraphs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikolettseas, S., Thomas, W. (eds.) ICALP 2009, Part II. LNCS, vol. 5556, pp. 508–520. Springer, Heidelberg (2009)
26. Koutsoupias, E., Pierrakos, G.: On the competitive ratio of online sampling auctions. *ACM Transactions on Economics and Computation* **1**(2), 10:1–10:10 (2013)
27. Kumar, R., Lattanzi, S., Vassilvitskii, S., Vattani, A.: Hiring a secretary from a poset. In: Proceedings of the 12th ACM Conference on Electronic Commerce, pp. 39–48 (2011)



28. Lavi, R., Nisan, N.: Competitive analysis of incentive compatible on-line auctions. In: Proceedings of the 2nd ACM Conference on Electronic Commerce, pp. 233–241 (2000)
29. Lindley, V.D.: Dynamic programming and decision theory. *Applied Statistics* **10**, 39–51 (1961)
30. Soto, A.J.: Matroid secretary problem in the random-assignment model. *SIAM Journal on Computing* **42**(1), 178–211 (2013)
31. Sun, X., Zhang, J., Zhang, J.: Solving Multi-choice Secretary Problem in Parallel: An Optimal Observation-Selection Protocol. [ArXiv:1405.5975v2](https://arxiv.org/abs/1405.5975v2) (2014)

# A Geometric Approach to Graph Isomorphism

Pawan Aurora and Shashank K. Mehta<sup>(✉)</sup>

Indian Institute of Technology, Kanpur 208016, India  
{paurora,skmehta}@cse.iitk.ac.in

**Abstract.** We present an integer linear program (IP), for the Graph Isomorphism (GI) problem, which has non-empty feasible solution if and only if the input pair of graphs are isomorphic. We study the polytope of the convex hull of the solution points of IP, denoted by  $\mathcal{B}^{[2]}$ . Exponentially many facets of this polytope are known. We show that in case of non-isomorphic pairs of graphs if a feasible solution exists for the linear program relaxation (LP) of the IP, then it violates a unique facet of  $\mathcal{B}^{[2]}$ . We present an algorithm for GI based on the solution of LP and prove that it detects non-isomorphism in polynomial time if the solution of the LP violates any of the known facets.

**Keywords:** Graph isomorphism problem · Linear programming · Polyhedral combinatorics

## 1 Introduction

The graph isomorphism problem (GI) is a well-studied computational problem: Formally, given two graphs  $G_1$  and  $G_2$  on  $n$  vertices, decide if there exists a bijection  $\sigma : V(G_1) \rightarrow V(G_2)$  such that  $\{u, v\} \in E(G_1)$  iff  $\{\sigma(u), \sigma(v)\} \in E(G_2)$ . Each such bijection is called an isomorphism. Without loss of generality, we assume that the vertices in both the graphs are labelled by integers  $1, \dots, n$ . Hence  $V(G_1) = V(G_2) = [n]$  and each bijection is a permutation of  $1, \dots, n$ . It remains one of the few problems that are unlikely to be NP-complete [1] and for which no polynomial time algorithm is known. The fastest known graph isomorphism algorithm for general graphs has running time  $2^{O(\sqrt{n \log n})}$  [4].

Several approaches to solve GI have been adopted. Most prominent of these has been the one that finds a canonical labeling of the vertices of the two graphs [3],[5]. For a comprehensive list of all the approaches there are some survey papers on the works published on this problem, such as [6].

Another problem of interest in the present context is Quadratic Assignment Problem (QAP) [8]. The QAP polytope is defined as the convex hull of all the feasible solutions to its linear formulation [9]. The polyhedral combinatorics of this polytope was studied by Volker Kaibel in his PhD thesis [7]. In the thesis he identifies a class of facets of this polytope and the dimension of its affine plane.

In this work we derive an integer linear program for graph isomorphism. Each (integer) solution of this program corresponds to one permutation. The

convex hull of these points is denoted by  $\mathcal{B}^{[2]}$  when both graphs are  $([n], \emptyset)$ . The polytope of the corresponding linear program (LP) is denoted by  $\mathcal{P}$ . We show that each maximally connected region of  $\mathcal{P} \setminus \mathcal{B}^{[2]}$  is separated from  $\mathcal{B}^{[2]}$  by only one facet. Hence in case of non-isomorphic graph pairs if the linear program is feasible, then the solutions violate exactly one facet of  $\mathcal{B}^{[2]}$ . Several facets of  $\mathcal{B}^{[2]}$  are already known and many new facets are identified in this paper.

We describe an algorithm which is based on the linear program. We show that if the linear program gives a feasible solution for non-isomorphic pair, then the algorithm correctly detects in polynomial time that the pair is non-isomorphic if the solution is separated from  $\mathcal{B}^{[2]}$  by any of the known facets. We show in [2] that there must be several additional facets of  $\mathcal{B}^{[2]}$  which are yet to be discovered. This is the reason that we cannot yet claim that this algorithm solves the graph isomorphism problem in polynomial time.

## 2 Integer Linear Program for GI

Define a second-order permutation matrix  $P_\sigma^{[2]}$  corresponding to a permutation  $\sigma$  as  $(P_\sigma^{[2]})_{ij,kl} = (P_\sigma)_{ij}(P_\sigma)_{kl}$ . We call the convex hull of the second-order permutation matrices, the *second-order Birkhoff polytope*  $\mathcal{B}^{[2]}$ . In [10] a completely positive formulation of *Quadratic Assignment Problem* (QAP) is given. The feasible region of this program is precisely  $\mathcal{B}^{[2]}$ , see theorem 3 in [10].

Let  $\mathcal{B}_{G_1 G_2}^{[2]}$  denote the convex hull of the  $P_\sigma^{[2]}$  where  $\sigma$  are the isomorphisms between  $G_1$  and  $G_2$ . If the graphs are non-isomorphic, then  $\mathcal{B}_{G_1 G_2}^{[2]} = \emptyset$ . Clearly  $\mathcal{B}^{[2]} = \mathcal{B}_{G_1 G_2}^{[2]}$  when  $G_1 = G_2 = ([n], \emptyset)$  or  $G_1 = G_2 = K_n$ .

**Observation 1.** *Given a pair of graphs, there exists a linear program (probably with exponentially many conditions) such that the feasible region of the program ( $\mathcal{B}_{G_1 G_2}^{[2]}$ ) is non-empty if and only if the graphs are isomorphic.*

Next we will develop an integer linear program such that the convex hull of its feasible points is  $\mathcal{B}_{G_1 G_2}^{[2]}$ . It is easy to verify that for every permutation  $\sigma$ ,  $Y = P_\sigma^{[2]}$  satisfies equations 1a-1d.

$$Y_{ij,kl} - Y_{kl,ij} = 0 \quad \forall i, j, k, l \tag{1a}$$

$$Y_{ij,il} = Y_{ji,li} = 0 \quad \forall i, \forall j \neq l \tag{1b}$$

$$\sum_k Y_{ij,kl} = \sum_k Y_{ij,lk} = Y_{ij,ij} \quad \forall i, j, l \tag{1c}$$

$$\sum_j Y_{ij,ij} = \sum_j Y_{ji,ji} = 1 \quad \forall i \tag{1d}$$

**Lemma 1.** *The solution plane,  $P$ , of equations 1a-1d is the affine plane spanned by  $P_\sigma^{[2]}$ 's, i.e.,  $P = \{\sum_\sigma \alpha_\sigma P_\sigma^{[2]} \mid \sum_\sigma \alpha_\sigma = 1\}$ .*

*Proof.* We will first show that the dimension of the solution plane is no more than  $n!/(2(n-4)!) + (n-1)^2 + 1$ .

In the following discussion we will split matrix  $Y$  into  $n^2$  non-overlapping sub-matrices of size  $n \times n$  which will be called *blocks*. The  $n$  blocks that contain the diagonal entries of  $Y$  will be called diagonal blocks. Note that  $Y_{ij,kl}$  is the  $jl$ -th entry of the  $ik$ -th block.

From the equation 1b, the off-diagonal entries of the diagonal blocks are zero. Assume that the first  $n - 1$  diagonal entries of each of the first  $n - 1$  diagonal blocks are given. Then all diagonal entries can be determined using equations 1d.

Consider any off diagonal block in the region above the main diagonal, other than the right most ( $n$ -th) block of that row. Note that the first entry of such a block will be  $Y_{r1,s1}$  where  $r < s < n$ . From the equation 1b we see that its diagonal entries are zero. The sum of the entries of any row of this block is same as the main diagonal entry of that row in  $Y$ , see equation 1c. Same holds for the columns from symmetry condition 1a. Hence by fixing all but one off-diagonal entries of the first principal sub-matrix of the block of size  $(n-1) \times (n-1)$ , we can fill in all the remaining entries. An exception to above is the second-last block of the  $(n-2)$ -th block-row (with first entry  $Y_{(n-2)1,(n-1)1}$ ). Here only the upper diagonal entries of the first principal sub-matrix of size  $(n-1) \times (n-1)$  are sufficient to determine all the remaining entries of that block. From equation 1c all the entries of the right most blocks can be determined. Lower diagonal entries of  $Y$  are determined by symmetry. Hence we see that the number of free variables is no more than  $(n-1)^2 + ((n-1)(n-2)-1)(2+\dots+(n-2)) + (n-1)(n-2)/2 = n!/(2(n-4)!) + (n-1)^2 + 1$ .

In [7] it is shown that the dimension of  $\mathcal{B}^{[2]}$  polytope is  $\frac{n!}{2(n-4)!} + (n-1)^2 + 1$ . This claim along with the result of the previous paragraph leads to the conclusion that equations 1a-1d define the affine plane spanned by the  $P_\sigma^{[2]}$ 's. □

**Corollary 1.**  $\mathcal{B}^{[2]}$  is a full dimensional polytope in  $P$ .

**Lemma 2.** The only 0/1 solutions of Equations 1a-1d are  $P_\sigma^{[2]}$ 's.

*Proof.* Let  $Y$  be a 0/1 solution of the above system of linear equations. Note that equations 1d and the non-negativity of the entries ensure that the diagonal of the solution is a vectorized doubly stochastic matrix. As the solution is a 0/1 matrix, the diagonal must be a vectorized permutation matrix, say  $P_\sigma$ . Then  $Y_{ij,ij} = (P_\sigma)_{ij}$ .

Equations 1c imply that  $Y_{ij,kl} = 1$  if and only if  $Y_{ij,ij} = 1$  and  $Y_{kl,kl} = 1$ . Equivalently,  $Y_{ij,kl} = Y_{ij,ij} \cdot Y_{kl,kl} = (P_\sigma)_{ij} \cdot (P_\sigma)_{kl} = (P_\sigma^{[2]})_{ij,kl}$ .

Equations 1a and 1b describe the remaining entries. □

Let  $G_1 = ([n], E_1)$  and  $G_2 = ([n], E_2)$  be simple graphs on  $n$  vertices each. Define a graph  $G = (V, E)$ , where  $V = [n] \times [n]$  and  $\{ij, kl\} \in E$  if either  $\{i, k\} \in E_1$  and  $\{j, l\} \in E_2$  or  $\{i, k\} \notin E_1$  and  $\{j, l\} \notin E_2$ .

**Corollary 2.** The only 0/1 solutions of equations 1a-1d and  $Y_{ij,kl} = 0 \forall \{ij, kl\} \notin E$ , are the  $P_\sigma^{[2]}$  where  $\sigma$  are the isomorphisms between  $G_1$  and  $G_2$ .

Corollary 2 gives the following integer program for GI.

$$\begin{aligned} \text{IP-GI: Find a point } Y & \\ \text{subject to } \mathbf{1a-1d} & \end{aligned} \tag{2a}$$

$$Y_{ij,kl} = 0 \quad , \{ij,kl\} \notin E \tag{2b}$$

$$Y_{ij,kl} \in \{0, 1\} \quad , \forall i, j, k, l$$

*Note.* This is a feasibility formulation of GI. To formulate an optimization program for GI, replace the conditions **1d** by  $\sum_i Y_{ij,ij} \leq 1$  and  $\sum_j Y_{ij,ij} \leq 1$ , and set the maximization objective function to be  $\sum_{i,j} Y_{ij,ij}$ . The solutions of IP-GI coincide with those solutions of the optimization version where the objective function evaluates to  $n$ .

The LP relaxation, LP-GI, is IP-GI with relaxed conditions on the variables. Here we only require that  $Y_{ij,kl} \geq 0$  for all  $i, j, k, l$ . The condition  $Y_{ij,kl} \leq 1$  is implicit for all  $i, j, k, l$ . Let  $\mathcal{P}_{G_1G_2}$  denote the feasible region of LP-GI. So  $\mathcal{B}_{G_1G_2}^{[2]} \subseteq \mathcal{P}_{G_1G_2}$ . Define  $\mathcal{P} = \mathcal{P}_{G_1G_2}$  where  $G_1 = G_2 = ([n], \emptyset)$  or  $G_1 = G_2 = K_n$ .  $\mathcal{P}$  is contained in the unit-cube  $\{0, 1\}^{n^2 \times n^2}$ , so it is a polytope. It is also contained in the plane  $P$ , hence it too is a full-dimensional polytope in that plane.

The following observations are in order.

**Observation 2.** *Graphs  $G_1, G_2$  are isomorphic if and only if the feasible region of LP-GI shares at least one point with  $\mathcal{B}^{[2]}$ .*

**Observation 3.** *The complete set of facets of  $\mathcal{P}$  is  $Y_{ij,kl} = 0 \forall i \neq k \forall j \neq l$ .*

**Observation 4.** *Vertices of  $\mathcal{B}^{[2]}$  (i.e.,  $P_\sigma^{[2]}$ ) are a subset of the vertices of  $\mathcal{P}$ .*

### 3 Facial Structure of $\mathcal{B}^{[2]}$

The feasible region of LP-GI for an isomorphic graph pair,  $G_1, G_2$ , will always contain at least one point from  $\mathcal{B}_{G_1G_2}^{[2]}$ . In case of non-isomorphic pair, either the feasible region will be empty or it will be confined to  $\mathcal{P} \setminus \mathcal{B}^{[2]}$ . While such solutions satisfy the non-negativity conditions, they occur on the wrong side of some of the facets of  $\mathcal{B}^{[2]}$ . We cannot include the corresponding inequalities into the linear program (even if we know them) and get an exact program for GI because they are exponentially large in number. It is easy to devise an algorithm for GI based on the present LP. Our goal is to identify the facets of  $\mathcal{B}^{[2]}$  and using the corresponding inequalities prove that this algorithm will take only polynomial time to detect that the entire LP feasible region is outside  $\mathcal{B}^{[2]}$ . Hence our first task is to identify all the  $\mathcal{B}^{[2]}$  facets. Exponentially many of these facets are already identified in the literature and we will identify exponentially many new facets.

We will represent a facet by an inequality  $f(x) \geq 0$  which defines the half space that contains the polytope and the plane  $f(x) = 0$  contains that facet. All

the known facets of  $\mathcal{B}^{[2]}$  are special instances of a general inequality

$$\sum_{ijkl} n_{ij} n_{kl} Y_{ij,kl} + (\beta - 1/2)^2 \geq (2\beta - 1) \sum_{ij} n_{ij} Y_{ij,ij} + 1/4 \tag{3}$$

where  $\beta \in \mathbb{Z}$  and  $n_{ij} \in \mathbb{Z}$  for all  $(ij)$ .

The first set of facets are the instances of this inequality where  $n_{i_0j_0} = n_{k_0l_0} = 1$  for some  $(i_0j_0) \neq (k_0l_0)$ , all other  $n_{ij} = 0$ , and  $\beta = 1$ .

**Theorem 5.**  $Y_{i_0j_0,k_0l_0} \geq 0$  defines a facet of  $\mathcal{B}^{[2]}$  for every  $i_0, j_0, k_0, l_0$  such that  $i_0 \neq k_0$  and  $j_0 \neq l_0$ .

The above theorem is proven in [7].

The next set of facets are due to  $\beta = n_{p_1q_1} = n_{p_2q_2} = n_{p_1q_2} = 1, n_{kl} = -1$ , and the rest of the  $n_{ij}$  are zero. Here  $p_1, p_2, k$  are any distinct set of indices. Similarly  $q_1, q_2, l$  are also any set of distinct indices.

**Theorem 6.** Inequality  $Y_{p_1q_1,kl} + Y_{p_2q_2,kl} + Y_{p_1q_2,kl} \leq Y_{kl,kl} + Y_{p_1q_1,p_2q_2}$  defines a facet of  $\mathcal{B}^{[2]}$ , where  $p_1, p_2, k$  are distinct and  $q_1, q_2, l$  are also distinct and  $n \geq 6$ .

The third set of facets is due to  $\beta = n_{i_1j_1} = \dots = n_{i_mj_m} = 1, n_{kl} = -1$  and the remaining  $n_{ij} = 0$ .

**Theorem 7.** Inequality  $Y_{i_1j_1,kl} + Y_{i_2j_2,kl} + \dots + Y_{i_mj_m,kl} \leq Y_{kl,kl} + \sum_{r \neq s} Y_{i_rj_r,i_sj_s}$ , defines a facet of  $\mathcal{B}^{[2]}$ , where  $i_1, \dots, i_m, k$  are all distinct and  $j_1, \dots, j_m, l$  are also distinct. In addition,  $n \geq 6, m \geq 3$ .

Theorems 6,7 appear with proof in [2] as Theorems 10,17 respectively.

The next two sets of facets are established in [7]. Let  $P_1$  and  $P_2$  be disjoint subsets of  $[n]$ . Similarly let  $Q_1$  and  $Q_2$  also be disjoint subsets of  $[n]$ . In these facets  $n_{ij} = 1$  if  $(ij) \in (P_1 \times Q_2) \cup (P_2 \times Q_1)$  and  $n_{ij} = -1$  if  $(ij) \in (P_1 \times Q_1) \cup (P_2 \times Q_2)$ . All other  $n_{ij}$  are zero. In the following case  $P_2 = Q_1 = \emptyset$ .

**Theorem 8.** [7, Definition 8.5] Following inequality defines a facet of  $\mathcal{B}^{[2]}$   
 $(\beta - 1) \sum_{(ij) \in P_1 \times Q_2} Y_{ij,ij} \leq \sum_{(ij) \neq (kl) \in P_1 \times Q_2, i < k} Y_{ij,kl} + (1/2)(\beta^2 - \beta)$   
 when  $\beta + 1 \leq |P_1|, |Q_2| \leq n - 3; |P_1| + |Q_2| \leq n - 3 + \beta; \beta \geq 2$ .

The next set of facets, with  $Q_1 = \emptyset$ , is given in the following theorem.

**Theorem 9.** [7, Definition 8.6] Following inequality defines a facet of  $\mathcal{B}^{[2]}$   
 $-(\beta - 1) \sum_{(ij) \in P_1 \times Q_2} Y_{ij,ij} + \beta \sum_{(ij) \in P_2 \times Q_2} Y_{ij,ij} + \sum_{(ij) \neq (kl) \in P_2 \times Q_2, i < k} Y_{ij,kl} + \sum_{(ij) \neq (kl) \in P_1 \times Q_2, i < k} Y_{ij,kl} - \sum_{(ij) \in P_1 \times Q_2, (kl) \in P_2 \times Q_2} Y_{ij,kl} + (1/2)(\beta^2 - \beta) \geq 0$   
 where the conditions on the parameters are as given in [7, Definition 8.6].

From Observations 3, 4 and Theorem 5 we have the following result.

**Theorem 10.** All facet defining planes of  $\mathcal{P}$  also define facets of  $\mathcal{B}^{[2]}$  and all vertices of  $\mathcal{B}^{[2]}$  are also vertices of  $\mathcal{P}$ . Besides, the dimensions of the two polytopes are same (both are full dimensional polytopes in plane  $P$ ).

Let the sets of vertices and the facet planes of  $\mathcal{B}^{[2]}$  be denoted by  $V(\mathcal{B}^{[2]})$  and  $F(\mathcal{B}^{[2]})$  respectively. Similarly denote the respective sets for  $\mathcal{P}$ .

Since  $V(\mathcal{B}^{[2]}) \subset V(\mathcal{P})$ ,  $\mathcal{B}^{[2]}$  is contained in  $\mathcal{P}$ .  $F(\mathcal{P})$  is a subset of  $F(\mathcal{B}^{[2]})$  and these facet planes are  $Y_{ij,kl} = 0$  planes so we refer to them as *zero-planes* and the corresponding facets as *trivial facets* of  $\mathcal{B}^{[2]}$ . Each facet  $X \in F(\mathcal{B}^{[2]}) \setminus F(\mathcal{P})$  partitions  $\mathcal{P}$  into two parts because the two polytopes are of the same dimension. One part contains  $\mathcal{B}^{[2]}$ . We will refer to the other part of  $\mathcal{P}$  by a *pocket*. Note that the points on the facet  $X$  do not belong to the pocket. The non- $P_\sigma^{[2]}$  vertices of  $\mathcal{P}$  do not belong to  $X$  since it is a facet of  $\mathcal{B}^{[2]}$  and it is the convex-hull of the  $P_\sigma^{[2]}$ s belonging to it. Clearly only non- $P_\sigma^{[2]}$  vertices must occur in the pockets. In the following lemma we will show that these pockets do not overlap, hence each vertex of  $V(\mathcal{P}) \setminus V(\mathcal{B}^{[2]})$  belongs to a unique pocket.

**Lemma 3.** *All pockets are disjoint.*

*Proof.* Polytope  $\mathcal{P}$  is the intersection of half spaces  $(Y_{ij,kl} \geq 0) \cap P$  in the underlying space  $P$ . Let  $I$  be the index set such that  $P$  is contained in the plane  $Y_{pq,rs} = 0$  for all  $(pq,rs) \in I$ . Hence a point  $Z \in P$  belongs to the interior of  $\mathcal{P}$  if and only if  $Z_{ij,kl} > 0$  for all  $(ij,kl) \notin I$ . Each pocket  $K$  is bounded by some zero planes,  $(Y_{ij,kl} = 0) \cap P$ , and one facet plane  $X \in F(\mathcal{B}^{[2]}) \setminus F(\mathcal{P})$ . So all pockets are full dimensional in  $P$ . By definition the points of  $X$  do not belong to  $K$ . Let  $\tilde{K}$  denote the interior of pocket  $K$ , which is a subset of the interior of  $\mathcal{P}$ .

Let  $K_1$  and  $K_2$  be two distinct pockets which are separated from  $\mathcal{B}^{[2]}$  by facet planes  $X_1$  and  $X_2$  respectively.

Let  $Z \in K_1 \cap K_2$ . Assume that  $Z \notin \tilde{K}_2$  but belongs to  $\tilde{K}_1$ . Then  $Z$  is a point on a face of  $K_2$  which is in the intersection of some zero planes. Let  $B(Z, \epsilon)$  be an infinitesimally small ball in  $P$  centered at  $Z$  with radius  $\epsilon$ . For small enough radius  $B(Z, \epsilon) \subset \tilde{K}_1$ . This is absurd because  $Z$  being on the intersection of some zero planes, there is a point in the ball which has at least one coordinate negative, which is not possible in the interior of  $K_1$ . Hence any zero plane bounding one pocket does not intersect the interior of the other pocket. We also know that  $X_1 \cap K_2 = X_2 \cap K_1 = \emptyset$ . Hence the interior of either pocket contains no boundary point of the other pocket.

Above argument shows that either  $\tilde{K}_1 = \tilde{K}_2$  or  $\tilde{K}_1 \cap \tilde{K}_2 = \emptyset$ . The former implies  $K_1 = K_2$ , which is not true. We deduce that the interiors of the pockets are disjoint.

Then the point  $Z$  must belong to a face in each pocket. These faces must be due to the intersection of some of the zero planes in  $P$ . Once again consider a ball  $B(Z, \epsilon)$  in  $P$ . Let  $B'(Z, \epsilon)$  denote the subset of the ball where all the non- $I$  coordinates are positive. Then for a small enough  $\epsilon$ ,  $B'(Z, \epsilon)$  must belong to the interior of  $\tilde{K}_1$  as well as of  $\tilde{K}_2$ . This implies that  $\tilde{K}_1 \cap \tilde{K}_2 \neq \emptyset$ . But that is not true as shown in the previous paragraph. So we conclude that  $K_1 \cap K_2$  must be empty. □

**Corollary 3.** *Each pocket is a maximal connected region of  $\mathcal{P} \setminus \mathcal{B}^{[2]}$ . And there is a one to one correspondence between facets of  $F(\mathcal{B}^{[2]}) \setminus F(\mathcal{P})$  and the pockets.*

Each vertex of  $V(\mathcal{P}) \setminus V(\mathcal{B}^{[2]})$  belongs to a unique pocket. So we have the following combinatorial result.

**Corollary 4.**  $|F(\mathcal{B}^{[2]}) \setminus F(\mathcal{P})| \leq |V(\mathcal{P}) \setminus V(\mathcal{B}^{[2]})|.$

**Lemma 4.** *The feasible region of LP-GI for non-isomorphic graphs, if non-empty, lies entirely in a unique pocket.*

*Proof.* Let the feasible region span multiple pockets. Let these include the pocket of the non-trivial facet  $X_1$ . By the definition of the feasible region, it cannot cross the boundary of  $\mathcal{P}$  hence it must cross  $X_1$  to enter another pocket. But  $X_1$  is a part of  $\mathcal{B}^{[2]}$  so the feasible region contains at least one point of  $\mathcal{B}^{[2]}$ . This implies that the graphs are isomorphic, falsifying the assumption.  $\square$

**Corollary 5.** *Every point in the feasible region of LP-GI for non-isomorphic graphs violates only one of the non-trivial facets of  $\mathcal{B}^{[2]}$ .*

### 3.1 There are More Facets

All the known facets of  $\mathcal{B}^{[2]}$  are instances of a general inequality  $\sum_{i,j,k,l} n_{ij}n_{kl} Y_{ij,kl} + (\beta - 1/2)^2 \geq (2\beta - 1) \sum_{ij} n_{ij} Y_{ij,ij} + 1/4$ . It is possible that there are more instances of this inequality which are also facets. Could there be facets of this polytope which are not the instances of this inequality. The answer to this question is in the affirmative. In [2, Section 5] we prove this claim.

In the following section we will show that a simple algorithm can be devised using LP-GI which can detect non-isomorphism in polynomial time, whenever the solution of a non-isomorphic pair belongs to a pocket corresponding to one of the facets described in this section. Hence it is essential to identify the remaining facets to determine how effective this algorithm is.

## 4 The Algorithm

We have seen that the feasible region of the linear program LP-GI strictly contains  $\mathcal{B}_{G_1 G_2}^{[2]}$ . Therefore if the feasible region of LP-GI is empty, then we can conclude that the graphs are non-isomorphic. But no conclusion can be derived in case LP-GI has a solution. In this section we propose an exact algorithm for graph isomorphism and show that if the solution is confined to any pocket which corresponds to any known facet (discussed in the previous section), then the time complexity of the algorithm is polynomial.

Let  $E$  denote the set of equations (2a-2b) and some additional equations of the form  $Y_{ij,kl} = 0$  or  $Y_{ij,kl} = 1$ . Let  $U$  denote the set of *live* variables which are not yet set to a fixed value (0 or 1) in  $E$ . Let  $LP(E, U)$  denote the linear program involving the equations  $E$  and inequalities  $Y_{ij,kl} \geq 0$  for each  $Y_{ijkl} \in U$ . Let  $SearchVar(E, U)$  be a subroutine which takes one variable  $Y_{ijkl}$  from  $U$  at a time and finds if the linear program is infeasible on setting this variable to 0 (then



returns  $(Y_{ij,kl}, 0)$  or on setting it to 1 (then returns  $(Y_{ij,kl}, 1)$ ). If the program is feasible for each assignment of each variable, then it returns  $(null, -1)$ .

Initially  $E_0$  denotes the system of equations (2a-2b) and  $U_0$  is the set of variables which are live with respect to equations (2a-2b). The main algorithm *GISolver* given in Algorithm 1 is called with parameters  $E_0$  and  $U_0$ .

```

Function: GISolver( $E, U$ )
if  $LP(E, U)$  is infeasible then
  | return false/* Graphs are non-isomorphic */
else
  | if  $LP(E, U)$  is feasible and  $U = \emptyset$  then
  | | return true/* Graphs are isomorphic */
  | else
  | |  $(x, r) := SearchVar(E, U);$ 
  | | if  $r = 1$  then
  | | | return  $GISolver(E \cup \{x = 0\}, U \setminus \{x\});$ 
  | | | else
  | | | | if  $r = 0$  then
  | | | | | return  $GISolver(E \cup \{x = 1\}, U \setminus \{x\});$ 
  | | | | | else
  | | | | | | Select a variable  $x$  from  $U$ ;
  | | | | | | return  $GISolver(E \cup \{x = 0\}, U \setminus \{x\}) \vee$ 
  | | | | | |  $GISolver(E \cup \{x = 1\}, U \setminus \{x\});$ 
  | | | | | end
  | | | end
  | | end
  | end
end

```

**Algorithm 1.** Algorithm for GI

If we view the space searched by *GISolver* as a tree with  $(E_0, U_0)$  as the root, then those nodes,  $(E, U)$ , have two child nodes where  $SearchVar(E, U)$  returns  $(null, -1)$ . Call them split nodes. All other internal nodes have one child each. Let there be at most  $\tau$  split nodes along any path from root to the leaves. Then the time complexity of this algorithm is  $O(p(n)2^\tau)$  where  $p(n)$  denotes a polynomial in  $n$ .

**4.1 Algorithm 1 Is Polynomial Time for Pockets of the Known Facets**

In this subsection we will show that if the feasible region of the linear program for a non-isomorphic pair is confined to a pocket corresponding to any of the known facets, then the algorithm will detect it in polynomial time. We will show that in these cases no split nodes will occur in the search-tree generated by the algorithm and hence  $\tau$  will be zero.

**Lemma 5.**  $\tau = 0$  when the feasible region lies in a pocket defined by any facet in Theorem 7 with  $m > 3$ .

*Proof.* Suppose the solution of a non-isomorphic pair is contained in the pocket of  $\sum_{r \in [m]} Y_{ir,jr,kl} \leq Y_{kl,kl} + \sum_{r < s \in [m]} Y_{ir,jr, is, js}$ , then the solutions will satisfy  $\sum_{r=1}^m Y_{ir,jr,kl} > Y_{kl,kl} + \sum_{r < s} Y_{ir,jr, is, js}$ . From Corollary 5, the solutions cannot violate any other facet. Let  $a$  be an arbitrary element of  $[m]$  and define  $S = [m] \setminus \{a\}$ . Then we have a facet due to  $\sum_{r \in S} Y_{ir,jr,kl} \leq Y_{kl,kl} + \sum_{r < s \in S} Y_{ir,jr, is, js}$  which must be satisfied by the solutions. Subtracting the second from the first we have  $Y_{ia,ja,kl} > \sum_{r \in S} Y_{ir,jr, ia, ja} \geq 0$ . The last inequality is due to the non-negativity condition in the linear program. This implies that when  $Y_{ia,ja,kl}$  will be set to zero in the algorithm, the linear program will declare it infeasible. Hence  $Y_{ia,ja,kl}$  will be set to 1. Since  $a$  is any arbitrary index, eventually  $Y_{ia,ja,kl}$  will be set to 1 for each  $a \in [m]$ . These will force  $Y_{kl,kl}$  and  $Y_{ir,jr, is, js} \forall r, s \in [m]$  to 1. Then the first inequality will be violated since the left hand side will be  $m$  but the right hand side will be  $1 + \binom{m}{2}$  where  $m \geq 4$ .  $\square$

**Lemma 6.**  $\tau = 0$  when the feasible region lies in a pocket defined by any facet in Theorem 8 with  $|P| > \beta + 1$  or  $|Q| > \beta + 1$ .

*Proof.* Assume that the inequality  $(\beta - 1) \sum_{(ij) \in P \times Q} Y_{ij,ij} \leq \sum_{(ij) \neq (kl) \in P \times Q, i < k} Y_{ij,kl} + (1/2)(\beta^2 - \beta)$  is violated and  $|P| > \beta + 1$ . We have

$$(\beta - 1) \sum_{(ij) \in P \times Q} Y_{ij,ij} > \sum_{(ij) \neq (kl) \in P \times Q, i < k} Y_{ij,kl} + (1/2)(\beta^2 - \beta) \tag{4}$$

Let  $i_0 \in P$  and  $j_0 \notin Q$ . Define  $P' = P \setminus \{i_0\}$ . Suppose during a call of *SearchVar* the algorithm forces  $Y_{i_0j_0, i_0j_0}$  to 1. Since  $P'$  and  $Q$  both have at least  $\beta + 1$  elements, the solution must satisfy the inequality

$$(\beta - 1) \sum_{(ij) \in P' \times Q} Y_{ij,ij} \leq \sum_{(ij) \neq (kl) \in P' \times Q, i < k} Y_{ij,kl} + (1/2)(\beta^2 - \beta). \tag{5}$$

(4) minus (5) gives  $(\beta - 1) \sum_{j \in Q} Y_{i_0j, i_0j} > \sum_{j \in Q} \sum_{(kl) \in P' \times Q} Y_{i_0j, kl}$ .

Since  $Y_{i_0j_0, i_0j_0} = 1$  where  $j_0 \notin Q$ ,  $\sum_{j \in Q} Y_{i_0j, i_0j} = 0$ . The non-negativity condition implies that the right-hand-side is non-negative so we conclude that  $0 > 0$ . As  $Y_{i_0j_0, i_0j_0} = 1$  renders the problem infeasible, the algorithm will set  $Y_{i_0j, i_0j} = 0$  for all  $j \notin Q$ . As  $i_0$  was an arbitrary element of  $P$ , eventually the algorithm will set  $Y_{ij,ij} = 0$  for all  $i \in P$  and all  $j \notin Q$ .

Next consider an arbitrary  $(i_0j_0) \in P \times Q$ . Suppose algorithm sets  $Y_{i_0j_0, i_0j_0} = 1$ . Let  $P' = P \setminus \{i_0\}$ . Then the violated inequality (4) reduces to  $(\beta - 1)(1 + \sum_{(ij) \in P' \times Q} Y_{ij,ij}) > \sum_{(ij) \neq (kl) \in P' \times Q, i < k} Y_{ij,kl} + \sum_{j \in Q} \sum_{(kl) \in P' \times Q} Y_{i_0j, kl} + \frac{\beta^2 - \beta}{2}$ .

Subtracting (5) from the above inequality gives  $(\beta - 1) > \sum_{j \in Q} \sum_{(kl) \in P' \times Q} Y_{i_0j, kl}$ . Since  $Y_{i_0j, kl} = 0$  for all  $j \neq j_0$ ,  $\sum_{j \notin Q} \sum_{(kl) \in P' \times Q} Y_{i_0j, kl} = 0$ . Adding this term to the right hand side of the inequality we get  $(\beta - 1) > \sum_{j \in [n]} \sum_{(kl) \in P' \times Q} Y_{i_0j, kl} = \sum_{(kl) \in P' \times Q} Y_{kl,kl}$ . From the first part of the proof,  $Y_{kl,kl} = 0$  for any  $k \in P$  and  $l \notin Q$ . So we have  $\sum_{(kl) \in P' \times Q} Y_{kl,kl} = |P'| > \beta + 1 - 1 = \beta$ . It reduces to infeasible  $\beta - 1 > \beta$ , which leads the algorithm to set  $Y_{i_0j_0, i_0j_0} = 0$ . Hence eventually  $Y_{ij,ij}$  is set to zero for all  $(ij) \in P \times Q$ . Combining with the fact that  $Y_{ij,ij} = 0$  for all  $i \in P, j \notin Q$ , we have  $1 = \sum_{j \in [n]} Y_{ij,ij} = 0$  for any  $i \in P$ . Hence algorithm will report non-isomorphic pair.  $\square$

**Lemma 7.**  $\tau = 0$  when the feasible region lies in a pocket defined by any facet in Theorem 8 with  $|P| = |Q| = \beta + 1$  and  $\beta > 2$ .

*Proof.* The violation of  $(\beta - 1) \sum_{(ij) \in P \times Q} Y_{ij,ij} \leq \sum_{(ij) \neq (kl) \in P \times Q, i < k} Y_{ij,kl} + (1/2)(\beta^2 - \beta)$  gives inequality (4), given in the last proof.

Let  $i_0 \in P$  and  $P' = P \setminus \{i_0\}$ . Then the solution must satisfy the facet with parameters  $P', Q, \beta - 1$ . So we have

$$(\beta - 2) \sum_{(ij) \in P' \times Q} Y_{ij,ij} \leq \sum_{(ij) \neq (kl) \in P' \times Q, i < k} Y_{ij,kl} + (1/2)((\beta - 1)^2 - (\beta - 1)) \tag{6}$$

(4) minus (6) gives

$$\sum_{(ij) \in P' \times Q} Y_{ij,ij} + (\beta - 1) \sum_{j \in Q} Y_{i_0j,i_0j} > \sum_{j \in Q} \sum_{(kl) \in P' \times Q} Y_{i_0j,kl} + (\beta - 1). \tag{7}$$

Since  $(\beta - 1) \sum_{j \in Q} Y_{i_0j,i_0j} = (\beta - 1) - (\beta - 1) \sum_{j \notin Q} Y_{i_0j,i_0j}$ , the inequality transforms to  $\sum_{(ij) \in P' \times Q} Y_{ij,ij} > (\beta - 1) \sum_{j \notin Q} Y_{i_0j,i_0j} + \sum_{j \in Q} \sum_{(kl) \in P' \times Q} Y_{i_0j,kl} = (|P'| - 1) \sum_{j \notin Q} Y_{i_0j,i_0j} + \sum_{j \in Q} \sum_{k \in P'} \sum_{l \in Q} Y_{i_0j,kl}$ , because  $\beta + 1 = |P| = |P'| + 1$ .

For  $Y$  is a solution of the LP,  $Y_{i_0j,i_0j} = \sum_{l \in [n]} Y_{i_0j,kl}$  for any  $k$ . So  $|P'| \sum_{j \notin Q} Y_{i_0j,i_0j} = \sum_{k \in P'} \sum_{j \notin Q} \sum_{l \in [n]} Y_{i_0j,kl}$ . Plugging this equation in the previous inequality we get  $\sum_{(ij) \in P' \times Q} Y_{ij,ij} > - \sum_{j \notin Q} Y_{i_0j,i_0j} + \sum_{k \in P'} \sum_{l \in [n]} \sum_{j \notin Q} Y_{i_0j,kl} + \sum_{k \in P'} \sum_{l \in Q} \sum_{j \in Q} Y_{i_0j,kl}$ . Combining the last two terms we get  $\sum_{(ij) \in P' \times Q} Y_{ij,ij} > - \sum_{j \notin Q} Y_{i_0j,i_0j} + \sum_{(kl) \in P' \times Q} \sum_{j \in [n]} Y_{i_0j,kl} = - \sum_{j \notin Q} Y_{i_0j,i_0j} + \sum_{k \in P'} \sum_{l \in Q} Y_{kl,kl}$ . It simplifies to  $\sum_{j \notin Q} Y_{i_0j,i_0j} > 0$ .

If the algorithm sets  $Y_{i_0j,i_0j} = 1$  for some  $j \in Q$ , then the above inequality will reduce to  $0 > 0$  making it infeasible. So eventually algorithm will set  $Y_{ij,ij} = 0$  for all  $(ij) \in P \times Q$ . This will make (4) infeasible and the algorithm will report that the graphs are non-isomorphic.  $\square$

Lemmas 6 and 7 lead to the following corollary.

**Corollary 6.**  $\tau = 0$  if the solution for a non-isomorphic pair is confined to a pocket defined by one of the facets given in Theorem 8 except when  $\beta = 2$  and  $|P| = |Q| = 3$ .

Lemma 5 and Corollary 6 imply that by adding additional inequalities corresponding to the base cases of facets in Theorems 7 and 8 to the constraints of LP-GI we can detect non-isomorphic graph pairs if their solution falls in any pocket defined by the facets given in Theorems 7 and 8. Moreover the facets in Theorem 6 can all be added to LP-GI. The additional inequalities will be polynomial in number ( $O(n^8)$ ), hence the modified LP-GI can be solved in polynomial time. Note that the facets given in Theorem 5 are already part of LP-GI.

**Lemma 8.**  $\tau = 0$  when the feasible region lies in a pocket defined by any facet in Theorem 9 subject to restrictions: (i)  $|P_1|, |P_2| \geq 3$ , (ii) if  $\beta \geq 0$  and  $\min\{|Q|, |P_1|\} \geq \beta + 1$  then  $|Q| + |P_1| + 3 \leq n + \beta$ , (iii) if  $\beta < 0$  and  $\min\{|Q|, |P_2|\} \geq 2 - \beta$  then  $|Q| + |P_2| + 3 \leq n + 1 - \beta$ .

*Proof.* Given that a 2-box facet  $(P_1, P_2, Q, \beta)$  is violated by the solution face, every solution point satisfies

$$\begin{aligned}
 & -(\beta - 1) \sum_{(ij) \in P_1 \times Q} Y_{ij,ij} + \beta \sum_{(ij) \in P_2 \times Q} Y_{ij,ij} + \sum_{(ij) \neq (kl) \in P_1 \times Q, i < k} Y_{ij,kl} \\
 & + \sum_{(ij) \neq (kl) \in P_2 \times Q, i < k} Y_{ij,kl} - \sum_{(ij) \in P_1 \times Q, (kl) \in P_2 \times Q} Y_{ij,kl} + \frac{\beta^2 - \beta}{2} < 0.
 \end{aligned} \tag{8}$$

Let  $i_0 \in P_1$  and  $i'_0 \in P_2$  be two arbitrary indices. Let  $P'_1 = P_1 \setminus \{i_0\}$  and  $P'_2 = P_2 \setminus \{i'_0\}$ . Then all the solutions must satisfy the inequality corresponding to the 2-box facet of  $(P'_1, P'_2, Q, \beta)$ . We have

$$\begin{aligned}
 & -(\beta - 1) \sum_{(ij) \in P'_1 \times Q} Y_{ij,ij} + \beta \sum_{(ij) \in P'_2 \times Q} Y_{ij,ij} + \sum_{(ij) \neq (kl) \in P'_1 \times Q, i < k} Y_{ij,kl} \\
 & + \sum_{(ij) \neq (kl) \in P'_2 \times Q, i < k} Y_{ij,kl} - \sum_{(ij) \in P'_1 \times Q, (kl) \in P'_2 \times Q} Y_{ij,kl} + \frac{\beta^2 - \beta}{2} \geq 0.
 \end{aligned} \tag{9}$$

Case 1: In the algorithm when  $Y_{i_0 j_0, i'_0 j'_0}$  is set to 1, where  $j_0, j'_0 \in Q, j_0 \neq j'_0$ , (8) minus (9) gives  $0 < 0$  which is absurd. Hence algorithm will set  $Y_{ij, i'j'} = 0$  for all  $i \in P_1, i' \in P_2, j, j' \in Q$ .

Case 2: When the algorithm sets  $Y_{i_0 j_0, i'_0 j'_0} = 1$ , where  $j_0 \notin Q, j'_0 \in Q$ . Then (8) minus (9) gives  $\beta + \sum_{(i,j) \in P'_2 \times Q} Y_{ij,ij} < 0$ , where we used the result of the previous case. Note that it is impossible if  $\beta \geq 0$ .

Case 3: When the algorithm sets  $Y_{i_0 l_0, i'_0 l'_0} = 1$ , where  $l_0 \in Q, l'_0 \notin Q$ . Then (8) minus (9) gives  $-(\beta - 1) + \sum_{(i,j) \in P'_1 \times Q} Y_{ij,ij} < 0$ , which is impossible if  $\beta < 0$ .

If  $\beta \geq 0$ , then combining the results of cases 1 and 2 we see that the algorithm sets  $Y_{ij,kl} = 0$  for all  $i \in P_1, k \in P_2, j \in [n], l \in Q$  which is same as setting  $Y_{ij,ij} = 0$  for all  $ij \in P_2 \times Q$ . Similarly we can see that if  $\beta < 0$ , then the algorithm will set  $Y_{ij,ij} = 0$  for all  $ij \in P_1 \times Q$ .

Plugging these values in inequality (8) we have following simplified cases

$$\beta < 0 : \beta \sum_{(ij) \in P_2 \times Q} Y_{ij,ij} + \sum_{(ij) \neq (kl) \in P_2 \times Q, i < k} Y_{ij,kl} + \frac{\beta^2 - \beta}{2} < 0. \tag{10}$$

$$\beta \geq 0 : -(\beta - 1) \sum_{(ij) \in P_1 \times Q} Y_{ij,ij} + \sum_{(ij) \neq (kl) \in P_1 \times Q, i < k} Y_{ij,kl} + \frac{\beta^2 - \beta}{2} < 0. \tag{11}$$

In the remainder we will prove that neither of these inequalities can be satisfied by the solution of the linear program in the subsequent phase of the algorithm. Hence, at this stage, the algorithm will find no solution and conclude that the graphs are non-isomorphic.

We will first consider inequality (11). If  $\beta \leq 1$  then clearly (11) is violated. So assume that  $\beta \geq 2$ . Next if  $|P_1|, |Q| \geq \beta + 1$ , then (11) implies that the 1-box facet corresponding to  $(P_1, Q, \beta)$  is violated. But that is impossible since the

solution can violate at most one non-trivial facet. That leaves us the case when  $\min\{|P_1|, |Q|\} \leq \beta$ .

First assume that  $|P_1| \leq |Q|$ . Consider the identity  $\sum_{ij \neq kl \in P_1 \times Q, i < k} Y_{ij,kl} = |P_1|(|P_1| - 1)/2 + \sum_{ij \neq kl \in P \times \bar{Q}, i < k} Y_{ij,kl} - (|P_1| - 1) \sum_{ij \in P_1 \times \bar{Q}} Y_{ij,ij}$ . Plugging into the inequality (11) gives  $-(\beta - 1) \sum_{ij \in P_1 \times Q} Y_{ij,ij} + |P_1|(|P_1| - 1)/2 + \sum_{ij \neq kl \in P_1 \times \bar{Q}, i < k} Y_{ij,kl} - (|P_1| - 1) \sum_{ij \in P_1 \times \bar{Q}} Y_{ij,ij} + \beta(\beta - 1)/2 < 0$ . But the left hand side of the above inequality is at least  $-(\beta - 1) \sum_{i \in P_1, j \in [n]} Y_{ij,ij} + \sum_{ij \neq kl \in P_1 \times \bar{Q}, i < k} Y_{ij,kl} + (\beta(\beta - 1) + |P_1|(|P_1| - 1))/2 \geq ((\beta - |P_1|)^2 - (\beta - |P_1|))/2 \geq 0$  since  $\sum_{i \in P_1, j \in [n]} Y_{ij,ij} = |P_1|$  and  $\beta, |P_1|$  are both integral. Hence we find that inequality (11) is impossible.

The case of  $|Q| \leq |P_1|$ , is handled similarly since  $P_1$  and  $Q$  have similar role. In case of inequality (10) we rewrite it by replacing  $\beta$  by  $-(\gamma - 1)$ . We get  $-(\gamma - 1) \sum_{(ij) \in P_2 \times Q} Y_{ij,ij} + \sum_{(ij) \neq (kl) \in P_2 \times Q, i < k} Y_{ij,kl} + (1/2)(\gamma^2 - \gamma) < 0$ . We can now use the same argument as above to establish that (10) is also impossible.  $\square$

**Theorem 11.** *Algorithm 1, using modified LP-GI, detects non-isomorphic graph pairs in polynomial time if the solution is confined to a pocket due to any of the facets described in Theorems 7,8 and a subset of facets described in Theorem 9.*

## 5 Conclusion

We have formulated GI as a geometric problem. The next challenge in establishing that GI is in class P lies in identifying the remaining facets of  $\mathcal{B}^{[2]}$  and proving that the corresponding  $\tau$  is at most  $O(\log n)$ . This does not contradict the fact that QAP is an NP-hard problem since in the present approach for GI, unlike QAP, the polytope of the linear program is not  $\mathcal{B}^{[2]}$ .

## References

1. Arvind, V., Torán, J.: Isomorphism testing: Perspective and open problems. Bulletin of the EATCS **86**, 66–84 (2005)
2. Aurora, P.K., Mehta, S.K.: New Facets of the QAP-Polytope. ArXiv e-prints (September 2014)
3. Babai, L., Kucera, L.: Canonical labelling of graphs in linear average time. In: FOCS, pp. 39–46 (1979)
4. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: STOC, pp. 171–183 (1983)
5. Babai, L., Erdős, P., Selkow, S.M.: Random graph isomorphism. SIAM J. Comput., 628–635 (1980)
6. Fortin, S.: The graph isomorphism problem. Technical report, University of Alberta (1996)
7. Kaibel, V.: Polyhedral Combinatorics of the Quadratic Assignment Problem. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Cologne (1997)
8. Koopmans, T.C., Beckmann, M.J.: Assignment problems and the location of economic activities. Technical report, Cowles Foundation, Yale University (1955)
9. Lawler, E.L.: The quadratic assignment problem. Management Science **9**(4), 586–599 (1963)
10. Povh, J., Rendl, F.: Copositive and semidefinite relaxations of the quadratic assignment problem. Discrete Optimization **6**(3), 231–241 (2009)

# Concentrated Hitting Times of Randomized Search Heuristics with Variable Drift

Per Kristian Lehre<sup>1</sup>(✉) and Carsten Witt<sup>2</sup>

<sup>1</sup> School of Computer Science, University of Nottingham,  
Nottingham NG8 1BB, UK

`PerKristian.Lehre@nottingham.ac.uk`

<sup>2</sup> DTU Compute, Technical University of Denmark, 2800 Lyngby, Denmark  
`cawi@dtu.dk`

**Abstract.** Drift analysis is one of the state-of-the-art techniques for the runtime analysis of randomized search heuristics (RSHs) such as evolutionary algorithms (EAs), simulated annealing etc. The vast majority of existing drift theorems yield bounds on the expected value of the hitting time for a target state, e. g., the set of optimal solutions, without making additional statements on the distribution of this time. We address this lack by providing a general drift theorem that includes bounds on the upper and lower tail of the hitting time distribution. The new tail bounds are applied to prove very precise sharp-concentration results on the running time of a simple EA on standard benchmark problems, including the class of general linear functions. The usefulness of the theorem outside the theory of RSHs is demonstrated by deriving tail bounds on the number of cycles in random permutations. All these results handle a position-dependent (variable) drift that was not covered by previous drift theorems with tail bounds. Moreover, our theorem can be specialized into virtually all existing drift theorems with drift towards the target from the literature. Finally, user-friendly specializations of the general drift theorem are given.

## 1 Introduction

Randomized search heuristics (RSHs) such as simulated annealing, EAs, ant colony optimization etc. are highly popular techniques in black-box optimization, the problem of optimizing a function with only oracle access to the function. These heuristics often imitate some natural process, and are rarely designed with analysis in mind. Their extensive use of randomness, such as in the mutation operator, render the underlying stochastic processes non-trivial. While the theory of RSHs is less developed than the theory of classical, randomized algorithms, significant progress has been made in the last decade [2, 11, 18]. This theory has mainly focused on the optimization time, which is the random variable  $T_{A,f}$  defined as the number of oracle accesses the heuristic  $A$  makes before the maximal argument of  $f$  is found. Most studies considered the expectation of  $T_{A,f}$ , however more information about the distribution of the optimization time is often needed. For example, the expectation can be deceiving when the

runtime distribution has a high variance. Also, for other performance measures, such as fixed-budget computation [5], tail bounds can be helpful.

Drift analysis is a central method for analyzing the hitting time of stochastic processes and was introduced to the analysis of simulated annealing as early as in 1988 [22]. Informally, it allows long-term properties of a discrete-time stochastic process  $(X_t)_{t \geq 0}$  to be inferred from properties of the one-step change  $\Delta_t := X_t - X_{t+1}$ . In the context of EAs, one has been particularly interested in the random variable  $T_a$  defined as the smallest  $t$  such that  $X_t \leq a$ . For example, if  $X_t$  represents the “distance” of the current solution in iteration  $t$  to an optimum, then  $T_0$  is the optimization time. Since its introduction to evolutionary computation by He and Yao in 2001 [10], drift analysis has been widely used to analyze the optimization time of EAs. Many drift theorems have been introduced, such as additive drift theorems [10], multiplicative drift [4, 6], variable drift [12, 17, 21], simplified drift [19, 20], and population drift [14]. Different assumptions and notation used in these theorems make it hard to abstract out a unifying statement.

Most drift theorems used in theory of RSHs relate to the expectation of the hitting time  $T_a$ , and there are fewer results about the tails  $\Pr(T_a > t)$  and  $\Pr(T_a < t)$ . From the simple observation that  $\Pr(T_a > t) \leq \Pr(\sum_{i=0}^t \Delta_i < a - X_0)$ , the problem is reduced to bounding the deviation of a sum of random variables. If the  $\Delta_t$  were independent and identically distributed, then one would be in the familiar scenario of Chernoff/Hoeffding-like bounds. The stochastic processes originating from RSHs are rarely so simple, in particular the  $\Delta_t$  are often dependent variables, and their distributions are not explicitly given. However, bounds on the form  $E(\Delta_t | X_t) \geq h(X_t)$  for some function  $h$  often hold. The drift is called *variable* when  $h$  is a non-constant function. The variable drift theorem provides bounds on the expectation of  $T_a$  given some conditions on  $h$ . However, there have been no general tail bounds from a variable drift condition. The only results in this direction seem to be the tail bounds for probabilistic recurrence relations from [13]; however, this scenario corresponds to the specific case of non-increasing  $X_t$ .

Our main contribution is a new, general drift theorem that provides sharp concentration results for the hitting time of stochastic processes with variable drift, along with concrete advice and examples how to apply it. The theorem is used to bound the tails of the optimization time of the well-known (1+1) EA [7] to the benchmark problems ONEMAX and LEADINGONES, as well as the class of linear functions, which is an intensively studied problem in the area [25]. Surprisingly, the results show that the distribution is highly concentrated around the expectation. The probability of deviating by an  $r$ -factor in lower order terms decreases exponentially with  $r$ . In a different application outside the theory of RSHs, we use drift analysis to analyze probabilistic recurrence relations and show that the number of cycles in a random permutation of  $n$  elements is sharply concentrated around the expectation  $\ln n$ . As a secondary contribution, we prove that our general drift theorem can be specialized into virtually all variants of drift theorems with drift towards the target (in particular, variable, additive, and multiplicative drift) that have been scattered over the literature on runtime analysis of RSHs. Unnecessary assumptions such as discrete or finite search spaces will be removed from these theorems.

This paper is structured as follows. Section 2 introduces notation and basics of drift analysis. Section 3 presents the general drift theorem with tail bounds and suggestions for user-friendly corollaries. Section 4 applies the tail bounds from our theorem. Sharp-concentration results on the running time of the (1+1) EA on ONEMAX, LEADINGONES and general linear functions are obtained. The application outside the theory of RSHs is described at the end of this section.

## 2 Preliminaries

We analyze time-discrete stochastic processes represented by a sequence of non-negative random variables  $(X_t)_{t \geq 0}$ . For example,  $X_t$  could represent a certain distance value of an RSH from an optimum. In particular,  $X_t$  might aggregate several different random variables realized by an RSH at time  $t$  into a single one. In contrast to existing drift theorems, we do not assume that the state space is discrete (e.g., all non-negative integers) or continuous but only demand that it is bounded, non-negative and includes the “target” 0.

We adopt the convention that the process should pass below some threshold  $a \geq 0$  (“minimizes” its state) and define the first hitting time  $T_a := \min\{t \mid X_t \leq a\}$ . If the actual process seeks to maximize its state, typically a straightforward mapping allows us to stick to the convention of minimization. In an important special case, we are interested in the hitting time  $T_0$  of target state 0. Note that  $T_a$  is a stopping time and that we assume that the stochastic process is adapted to some filtration  $(\mathcal{F}_t)_{t \geq 0}$ , such as its natural filtration  $\sigma(X_0, \dots, X_t)$ .

The expected one-step change  $\delta_t := E(X_t - X_{t+1} \mid \mathcal{F}_t)$  for  $t \geq 0$  is called *drift*. Note that  $\delta_t$  is in general a random variable as the outcomes of  $X_0, \dots, X_t$  are random. Suppose we manage to bound  $\delta_t$  from below by some  $\delta^* > 0$  for all possible outcomes of  $\delta_t$ , where  $t < T$ . Then we know that the process decreases its state (“progresses towards 0”) in expectation by at least  $\delta^*$  in every step, and the additive drift theorem (see Theorem 1 below) will provide a bound on  $T_0$  that only depends on  $X_0$  and  $\delta^*$ . In fact, the very natural-looking result  $E(T_0 \mid X_0) \leq X_0/\delta^*$  will be obtained. However, bounds on the drift might be more complicated. For example, a bound on  $\delta_t$  might depend on  $X_t$  or states at even earlier points of time, e.g., if the progress decreases as the current state decreases. This is often the case in applications to EAs. However, for such algorithms the whole “history” is rarely needed. Simple EAs and other RSHs are Markov processes such that often  $\delta_t = E(X_t - X_{t+1} \mid X_t)$  for an appropriate  $X_t$ .

We now present the first drift theorem for additive drift. It is based on [10], from which we removed the unnecessary assumptions that the search space is discrete and the Markov property. We only demand a bounded state space.

**Theorem 1 (Additive Drift, following [10]).** *Let  $(X_t)_{t \geq 0}$ , be a stochastic process over a bounded state space  $S \subseteq \mathbb{R}_0^+$ . Assume that  $E(T_0 \mid X_0) < \infty$ . Then:*

- (i) *If  $E(X_t - X_{t+1}; X_t > 0 \mid \mathcal{F}_t) \geq \delta_u$  then  $E(T_0 \mid X_0) \leq \frac{X_0}{\delta_u}$ .*
- (ii) *If  $E(X_t - X_{t+1}; X_t > 0 \mid \mathcal{F}_t) \leq \delta_\ell$  then  $E(T_0 \mid X_0) \geq \frac{X_0}{\delta_\ell}$ .*



Summing up, additive drift is concerned with the simple scenario that there is a progress of at least  $\delta_u$  from all non-optimal states towards the target in (i) and a progress of at most  $\delta_\ell$  in (ii). Since the  $\delta$ -values are independent of  $X_t$ , one has to use the worst-case drift over all non-optimal  $X_t$ . This might lead to very bad bounds on the first hitting time, which is why more general theorems (as mentioned in the introduction) were developed. Interestingly, these more general theorems are often proved based on Theorem 1 using an appropriate mapping (sometimes called *potential function*, *distance function* or *drift function*) from the original state space to a new one. Informally, the mapping “smoothes out” position-dependent drift into an (almost) position-independent drift.

### 3 General Drift Theorem

In this section, we present our general drift theorem. As pointed out in the introduction, we strive for a very general statement, which is partly at the expense of simplicity. More user-friendly specializations will be given later. Nevertheless, the underlying idea of the complicated-looking general theorem is the same as in all drift theorems. We look into the one-step drift  $E(X_t - X_{t+1} \mid \mathcal{F}_t)$  and assume we have a (upper or lower) bound  $h(X_t)$  on the drift, which (possibly heavily) depends on  $X_t$ . Based on  $h$ , we define a new function  $g$  (see Remark 1), with the aim of “smoothing out” the dependency, and the drift w. r. t.  $g$  (formally,  $E(g(X_t) - g(X_{t+1}) \mid \mathcal{F}_t)$ ) is analyzed. Statements (i) and (ii) of the following Theorem 2 provide bounds on  $E(T_0)$  based on the drift w. r. t.  $g$ . In fact,  $g$  can be defined in a very similar way as in existing variable-drift theorems [12, 17, 21], such that Statements (i) and (ii) can be understood as generalized variable drift theorems for upper and lower bounds on the expected hitting time, respectively.

Statements (iii) and (iv) are concerned with tail bounds on the hitting time. Here moment-generating functions (mgfs.) of the drift w. r. t.  $g$  come into play, formally  $E(e^{-\lambda(g(X_t) - g(X_{t+1}))} \mid \mathcal{F}_t)$  is bounded. Again for generality, bounds on the mgf. may depend on the point of time  $t$ , as captured by the bounds  $\beta_u(t)$  and  $\beta_\ell(t)$ . We will see an example in Section 4 where the mapping  $g$  smoothes out the position-dependent drift into a (nearly) position-independent and time-independent drift, while the mgf. of the drift w. r. t.  $g$  still heavily depends on the current point of time  $t$  (and indirectly on the position expected at this time).

It can be shown (see proofs in [16]) that our drift theorem generalizes virtually all existing drift theorems, including the variable drift theorems for upper [12, 17, 21] and lower bounds [3], a non-monotone variable drift theorem [8], and multiplicative drift theorems [4, 25]. It can also be shown (see proofs in [16]) that our theorem generalizes fitness-level theorems [23, 24, 26], another well-known technique in the analysis of randomized search heuristics.

*Remark 1.* If for some function  $h: [x_{\min}, x_{\max}] \rightarrow \mathbb{R}^+$  where  $1/h(x)$  is integrable on  $[x_{\min}, x_{\max}]$ , either  $E(X_t - X_{t+1} \mid \mathcal{F}_t) \geq h(X_t)$  or  $E(X_t - X_{t+1} \mid \mathcal{F}_t) \leq h(X_t)$  hold, it is recommended to define the function  $g$  in Theorem 2 as  $g(x) := \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^x \frac{1}{h(y)} dy$  for  $x \geq x_{\min}$  and  $g(0) := 0$ .

**Theorem 2 (General Drift Theorem<sup>1</sup>).** *Let  $(X_t)_{t \geq 0}$ , be a stochastic process, adapted to a filtration  $(\mathcal{F}_t)_{t \geq 0}$ , over some state space  $S \subseteq \{0\} \cup [x_{\min}, x_{\max}]$ , where  $x_{\min} \geq 0$ . Let  $g: \{0\} \cup [x_{\min}, x_{\max}] \rightarrow \mathbb{R}^{\geq 0}$  be any function such that  $g(0) = 0$ , and  $0 < g(x) < \infty$  for all  $x \in [x_{\min}, x_{\max}]$ . Let  $T_a = \min\{t \mid X_t \leq a\}$  for  $a \in \{0\} \cup [x_{\min}, x_{\max}]$ . Then:*

- (i) *If  $E(g(X_t) - g(X_{t+1}); X_t \geq x_{\min} \mid \mathcal{F}_t) \geq \alpha_u$  for some  $\alpha_u > 0$  then  $E(T_0 \mid X_0) \leq \frac{g(X_0)}{\alpha_u}$ .*
- (ii) *If  $E(g(X_t) - g(X_{t+1})); X_t \geq x_{\min} \mid \mathcal{F}_t) \leq \alpha_\ell$  for some  $\alpha_\ell > 0$  then  $E(T_0 \mid X_0) \geq \frac{g(X_0)}{\alpha_\ell}$ .*
- (iii) *If there exists  $\lambda > 0$  and a function  $\beta_u: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  such that  $E(e^{-\lambda(g(X_t) - g(X_{t+1}))}; X_t > a \mid \mathcal{F}_t) \leq \beta_u(t)$  then  $\Pr(T_a > t \mid X_0) < \left(\prod_{r=0}^{t-1} \beta_u(r)\right) \cdot e^{\lambda(g(X_0) - g(a))}$  for  $t > 0$ .*
- (iv) *If there exists  $\lambda > 0$  and a function  $\beta_\ell: \mathbb{N}_0 \rightarrow \mathbb{R}^+$  such that  $E(e^{\lambda(g(X_t) - g(X_{t+1}))}; X_t > a \mid \mathcal{F}_t) \leq \beta_\ell(t)$  then  $\Pr(T_a < t \mid X_0 > a) \leq \left(\sum_{s=1}^{t-1} \prod_{r=0}^{s-1} \beta_\ell(r)\right) \cdot e^{-\lambda(g(X_0) - g(a))}$  for  $t > 0$ .  
If additionally the set of states  $S \cap \{x \mid x \leq a\}$  is absorbing, then  $\Pr(T_a < t \mid X_0 > a) \leq \left(\prod_{r=0}^{t-1} \beta_\ell(r)\right) \cdot e^{-\lambda(g(X_0) - g(a))}$ .*

**Special Cases of (iii) and (iv).** If  $E(e^{-\lambda(g(X_t) - g(X_{t+1}))}; X_t > a \mid \mathcal{F}_t) \leq \beta_u$  for some time-independent  $\beta_u$ , then Statement (iii) simplifies to  $\Pr(T_a > t \mid X_0) < \beta_u^t \cdot e^{\lambda(g(X_0) - g(a))}$ ; similarly for Statement (iv). The proof of our main theorem is not too complicated and can be found in [16]. The tail bounds in (iii) and (iv) are obtained by the exponential method (a generalized Chernoff bound), which idea is also implicit in [9].

Given some assumptions on the “drift” function  $h$  that typically hold, Theorem 2 can be simplified.

**Corollary 1.** *Let  $(X_t)_{t \geq 0}$ , be a stochastic process, adapted to a filtration  $(\mathcal{F}_t)_{t \geq 0}$ , over some state space  $S \subseteq \{0\} \cup [x_{\min}, x_{\max}]$ , where  $x_{\min} \geq 0$ . Let  $h: [x_{\min}, x_{\max}] \rightarrow \mathbb{R}^+$  be a function such that  $1/h(x)$  is integrable and  $h(x)$  differentiable on  $[x_{\min}, x_{\max}]$ . Then the following hold for the first hitting time  $T := \min\{t \mid X_t = 0\}$ .*

- (i) *If  $E(X_t - X_{t+1}; X_t \geq x_{\min} \mid \mathcal{F}_t) \geq h(X_t)$  and  $\frac{d}{dx}h(x) \geq 0$ , then  $E(T \mid X_0) \leq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(y)} dy$ .*
- (ii) *If  $E(X_t - X_{t+1}; X_t \geq x_{\min} \mid \mathcal{F}_t) \leq h(X_t)$  and  $\frac{d}{dx}h(x) \leq 0$ , then  $E(T \mid X_0) \geq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(y)} dy$ .*
- (iii) *If  $E(X_t - X_{t+1}; X_t \geq x_{\min} \mid \mathcal{F}_t) \geq h(X_t)$  and  $\frac{d}{dx}h(x) \geq \lambda$  for some  $\lambda > 0$ , then  $\Pr(T > t \mid X_0) < \exp\left(-\lambda\left(t - \frac{x_{\min}}{h(x_{\min})} - \int_{x_{\min}}^{X_0} \frac{1}{h(y)} dy\right)\right)$ .*
- (iv) *If  $E(X_t - X_{t+1}; X_t \geq x_{\min} \mid \mathcal{F}_t) \leq h(X_t)$  and  $\frac{d}{dx}h(x) \leq -\lambda$  for some  $\lambda > 0$ , then  $\Pr(T < t \mid X_0 > 0) < \frac{e^{\lambda t} - e^\lambda}{e^\lambda - 1} \exp\left(-\frac{\lambda x_{\min}}{h(x_{\min})} - \int_{x_{\min}}^{X_0} \frac{\lambda}{h(y)} dy\right)$ .*

<sup>1</sup> Proofs omitted from this paper due to space restrictions can be found in [16].

Condition (iii) and (iv) of Theorem 2 involve an mgf., which may be tedious to compute. Inspired by [9] and [15], we show that bounds on the mgfs. follow from more user-friendly conditions based on stochastic dominance between random variables, here denoted by  $\prec$ .

**Theorem 3.** *Let  $(X_t)_{t \geq 0}$ , be a stochastic process, adapted to a filtration  $(\mathcal{F}_t)_{t \geq 0}$ , over some state space  $S \subseteq \{0\} \cup [x_{\min}, x_{\max}]$ , where  $x_{\min} \geq 0$ . Let  $h: [x_{\min}, x_{\max}] \rightarrow \mathbb{R}^+$  be a function such that  $1/h(x)$  is integrable on  $[x_{\min}, x_{\max}]$ . Suppose there exist a random variable  $Z$  and some  $\lambda > 0$  such that  $|\int_{X_{t+1}}^{X_t} 1/h(x) dx| \prec Z$  for  $X_t \geq x_{\min}$  and  $E(e^{\lambda Z}) = D$  for some  $D > 0$ . Then the following two statements hold for the first hitting time  $T := \min\{t \mid X_t = 0\}$ .*

- (i) *If  $E(X_t - X_{t+1} \mid X_t \geq x_{\min} \mid \mathcal{F}_t) \geq h(X_t)$  then for any  $\delta > 0$ , and  $\eta := \min\{\lambda, \delta\lambda^2/(D - 1 - \lambda)\}$  and  $t > 0$  it holds that  $\Pr(T > t \mid X_0) \leq \exp\left(\eta\left(\int_{x_{\min}}^{X_0} 1/h(x) dx - (1 - \delta)t\right)\right)$ .*
- (ii) *If  $E(X_t - X_{t+1} \mid X_t \geq x_{\min} \mid \mathcal{F}_t) \leq h(X_t)$  then for any  $\delta > 0$ ,  $\eta := \min\{\lambda, \delta\lambda^2/(D - 1 - \lambda)\}$  and  $t > 0$  it holds*

$$\Pr(T < t \mid X_0) \leq \exp\left(\eta\left((1 + \delta)t - \int_{x_{\min}}^{X_0} 1/h(x) dx\right)\right) \frac{1}{\eta(1 + \delta)}.$$

*If state 0 is absorbing then*

$$\Pr(T < t \mid X_0) \leq \exp\left(\eta((1 + \delta)t - \int_{x_{\min}}^{X_0} 1/h(x) dx)\right).$$

### 4 Applications of the Tail Bounds

As our main contribution, we show that the general drift theorem (Theorem 2), together with the function  $g$  defined explicitly in Remark 1 in terms of the one-step drift, constitute a very general and precise tool for analysis of stochastic processes. In particular, it provides very sharp tail bounds on the running time of randomized search heuristics which were not obtained before by drift analysis. Virtually all existing drift theorems dealing with drift towards a target can be phrased as special cases of the general drift theorem (see [16]). It also provides tail bounds on random recursions, such as those in analysis of random permutations (see Section 4.2).

We first give sharp tail bounds on the optimization time of the (1+1) EA which maximizes pseudo-Boolean functions  $f: \{0, 1\}^n \rightarrow \mathbb{R}$ . We consider classical benchmark problems from the theory of RSHs. Despite their simplicity, their analysis has turned out surprisingly difficult and research is still ongoing.

---

**Algorithm 1.** (1+1) Evolutionary Algorithm (EA)

---

Choose uniformly at random an initial bit string  $x_0 \in \{0, 1\}^n$ .  
**for**  $t := 0$  **to**  $\infty$  **do**  
    Create  $x'$  by flipping each bit in  $x_t$  independently with probability  $1/n$  (*mutation*).  
     $x_{t+1} := x'$  if  $f(x') \geq f(x_t)$ , and  $x_{t+1} := x_t$  otherwise (*selection*).  
**end for**

---

**4.1 OneMax, Linear Functions and LeadingOnes**

A simple pseudo-Boolean function is given by  $\text{ONEMAX}(x_1, \dots, x_n) = x_1 + \dots + x_n$ . It is included in the class of so-called linear functions  $f(x_1, \dots, x_n) = w_1x_1 + \dots + w_nx_n$ , where  $w_i \in \mathbb{R}$  for  $1 \leq i \leq n$ . We start by deriving very precise bounds on first the expected optimization time of the (1+1) EA on ONEMAX and then prove tail bounds. The lower bounds obtained will imply results for all linear functions. Note that in [3], it was already shown using variable drift analysis that the expected optimization time of the (1+1) EA on ONEMAX is at most  $en \ln n - c_1n + O(1)$  and at least  $en \ln n - c_2n$  for certain constants  $c_1, c_2 > 0$ . The constant  $c_2$  is not made explicit in [3], whereas the constant  $c_1$  is stated as 0.369. However, unfortunately this value is due to a typo in the very last line of the proof –  $c_1$  should have been 0.1369 instead. We correct this mistake in a self-contained proof. Furthermore, we improve the lower bound using variable drift. To this end, we use the following bound on the drift.

**Lemma 1.** *Let  $X_t$  denote the number of zeros of the current search point of the (1+1) EA on ONEMAX. Then*

$$\left(1 - \frac{1}{n}\right)^{n-x} \frac{x}{n} \leq E(X_t - X_{t+1} \mid X_t = x) \leq \left(\left(1 - \frac{1}{n}\right) \left(1 + \frac{x}{(n-1)^2}\right)\right)^{n-x} \frac{x}{n}.$$

**Theorem 4.** *The expected optimization time of the (1+1) EA on ONEMAX is at most  $en \ln n - 0.1369n + O(1)$  and at least  $en \ln n - 7.81791n - O(\log n)$ .*

Knowing the expected optimization time precisely, we now turn to our main new contribution, i. e., to derive sharp bounds. Note that the following upper concentration inequality in Theorem 5 is not new but is already implicit in the work on multiplicative drift analysis by [6]. In fact, a very similar upper bound is even available for all linear functions [25]. By contrast, the lower concentration inequality is a novel and non-trivial result.

**Theorem 5.** *The optimization time of the (1+1) EA on ONEMAX is at least  $en \ln n - cn - ren$ , where  $c$  is a constant, with probability at least  $1 - e^{-r/2}$  for any  $r \geq 0$ . It is at most  $en \ln n + ren$  with probability at least  $1 - e^{-r}$ .*

We only consider the lower tail. The aim is to prove it using Theorem 2.(iv), which includes a bound on the moment-generating function of the drift of  $g$ . We first set up the  $h$  (and thereby the  $g$ ) used for our purposes. Obviously,  $x_{\min} := 1$ .

**Lemma 2.** Consider the (1+1) EA on ONEMAX and let the random variable  $X_t$  denote the current number of zeros at time  $t \geq 0$ . Then  $h(x) := \exp(-1 + 2\lceil x \rceil/n) \cdot (\lceil x \rceil/n) \cdot (1 + c^*/n)$ , where  $c^* > 0$  is a sufficiently large constant, satisfies the condition  $E(X_t - X_{t+1} \mid X_t = i) \leq h(i)$  for  $i \in \{1, \dots, n\}$ . Moreover, define  $g(i) := x_{\min}/h(x_{\min}) + \int_{x_{\min}}^i 1/h(y) dy$  and  $\Delta_t := g(X_t) - g(X_{t+1})$ . Then  $g(i) = \sum_{j=1}^i 1/h(j)$  and  $\Delta_t \leq \sum_{j=X_{t+1}+1}^{X_t} e^{1-2X_{t+1}/n} \cdot (n/j)$ .

The next lemma provides a bound on the mgf. of the drift of  $g$ , which will depend on the current state. Later, the state will be estimated based on the current point of time, leading to a time-dependent bound on the mgf. Note that we do not need the whole natural filtration based on  $X_0, \dots, X_t$  but only  $X_t$  since we are dealing with a Markov chain.

**Lemma 3.** Let  $\lambda := 1/(en)$  and  $i \in \{1, \dots, n\}$ . Then  $E(e^{\lambda \Delta_t} \mid X_t = i) \leq 1 + \lambda + 2\lambda/i + o(\lambda/\log n)$ .

The bound on the mgf. of  $\Delta_t$  derived in Lemma 3 is particularly large for  $i = O(1)$ , i. e., if the current state  $X_t$  is small. If  $X_t = O(1)$  held during the whole optimization process, we could not prove the lower tail in Theorem 5 from the lemma. However, it is easy to see that  $X_t = i$  only holds for an expected number of at most  $en/i$  steps. Hence, most of the time the term  $2\lambda/i$  is negligible, and the time-dependent  $\beta_\ell(t)$ -term from Theorem 2.(iv) comes into play. We make this precise in the following proof, where we iteratively bound the probability of the process being at “small” states.

*Proof (of Theorem 5, Lower Tail).* With overwhelming probability  $1 - 2^{-\Omega(n)}$ ,  $X_0 \geq (1 - \epsilon)n/2$  for an arbitrarily small constant  $\epsilon > 0$ , which we assume to happen. We consider phases in the optimization process. Phase 1 starts with initialization and ends before the first step where  $X_t < e^{\frac{\ln n - 1}{2}} = \sqrt{n} \cdot e^{-1/2}$ . Phase  $i$ , where  $i > 1$ , follows Phase  $i - 1$  and ends before the first step where  $X_t < \sqrt{n} \cdot e^{-i/2}$ . Obviously, the optimum is not found before the end of Phase  $\ln(n)$ ; however, this does not tell us anything about the optimization time yet.

We say that Phase  $i$  is *typical* if it does not end before time  $eni - 1$ . We will prove inductively that the probability of one of the first  $i$  phases not being typical is at most  $c'e^{\frac{i}{2}}/\sqrt{n} = c'e^{\frac{i - \ln n}{2}}$  for some constant  $c' > 0$ . This implies the theorem since an optimization time of at least  $en \ln n - cn - ren$  is implied by the event that Phase  $\ln n - \lceil r - c/e \rceil$  is typical, which has probability at least  $1 - c'e^{\frac{-r+c/e+1}{2}} = 1 - e^{\frac{-r}{2}}$  for  $c = e(2 \ln c' + 1)$ .

Fix some  $k > 1$  and assume for the moment that all first  $k - 1$  phases are typical. Then for  $1 \leq i \leq k - 1$ , we have  $X_t \geq \sqrt{n}e^{-i/2}$  in Phase  $i$ , i. e., when  $en(i-1) \leq t \leq eni - 1$ . We analyze the event that additionally Phase  $k$  is typical, which subsumes the event  $X_t \geq \sqrt{n}e^{-k/2}$  throughout Phase  $k$ . According to Lemma 3, we get in Phase  $i$ , where  $1 \leq i \leq k$ ,

$$E(e^{\lambda \Delta_t} \mid X_t) \leq 1 + \lambda + 2\lambda e^{i/2}/\sqrt{n} + o(\lambda/\ln n) \leq e^{\lambda + \frac{2\lambda e^{i/2}}{\sqrt{n}} + o(\frac{\lambda}{\ln n})}.$$

The expression now depends on the time only, therefore for  $\lambda := 1/(en)$

$$\prod_{t=0}^{enk-1} E(e^{\lambda \Delta_t} \mid X_0) \leq e^{\lambda enk + \frac{2\lambda en}{\sqrt{n}} \sum_{i=1}^k e^{i/2} + enk \cdot o(\frac{\lambda}{\ln n})} \leq e^{k + \frac{6ek/2}{n\sqrt{n}} + o(1)} \leq e^{k+o(1)},$$

where we used that  $k \leq \ln n$ . From Theorem 2.(iv) for  $a = \sqrt{n}e^{-k/2}$  and  $t = enk - 1$  we obtain  $\Pr(T_a < t) \leq e^{k+o(1) - \lambda(g(X_0) - g(\sqrt{n}e^{-k/2}))}$ . From the proof of Theorem 4, lower bound part, we already know that  $g(X_0) \geq en \ln n - c'n$  for some constant  $c' > 0$  (which is assumed large enough to subsume the  $-O(\log n)$  term). Moreover,  $g(x) \leq en(\ln x + 1)$  according to Lemma 2. We get

$$\Pr(T_a < t) \leq e^{k+o(1) - \ln n + O(1) - k/2 + (\ln n)/2} = e^{\frac{k - \ln n + O(1)}{2}} = c'''e^{k/2}/\sqrt{n},$$

for some sufficiently large constant  $c''' > 0$ , which proves the bound on the probability of Phase  $k$  not being typical (without making statements about the earlier phases). The probability that all phases up to and including Phase  $k$  are typical is at least  $1 - (\sum_{i=1}^k c'''e^{i/2})/\sqrt{n} \geq 1 - c'e^{k/2}/\sqrt{n}$  for a constant  $c' > 0$ . □

We now deduce a concentration inequality w. r. t. linear functions, i. e., functions of the kind  $f(x_1, \dots, x_n) = w_1x_1 + \dots + w_nx_n$ , where  $w_i \neq 0$ .

**Theorem 6.** *The optimization time of the (1+1) EA on an arbitrary linear function with non-zero weights is at least  $en \ln n - cn - ren$ , where  $c$  is a constant, with probability at least  $1 - e^{-r/2}$  for any  $r \geq 0$ . It is at most  $en \ln n + (1+r)en + O(1)$  with probability at least  $1 - e^{-r}$ .*

*Proof.* The upper tail is proved in Theorem 5.1 in [25]. The lower bound follows from the lower tail in Theorem 5 in conjunction with the fact that the optimization time within the class of linear functions is stochastically smallest for ONEMAX (Theorem 6.2 in [25]). □

Finally, we consider  $\text{LEADINGONES}(x_1, \dots, x_n) := \sum_{i=1}^n \prod_{j=1}^i x_j$ , another intensively studied standard benchmark problem from the analysis of RSHs. Tail bounds on the optimization time of the (1+1) EA on LEADINGONES were derived in [5]. This result represents a fundamentally new contribution, but suffers from the fact that it depends on a very specific structure and closed formula for the optimization time. Using a simplified version of Theorem 2 (see Theorem 3), it is possible to prove similarly strong tail bounds without needing this exact formula. As in [5], we are interested in a more general statement. Let  $T(a)$  be the number of steps until an LEADINGONES-value of at least  $a$  is reached, where  $0 \leq a \leq n$ . Let  $X_t := \max\{0, a - \text{LEADINGONES}(x_t)\}$  be the distance from the target  $a$  at time  $t$ . Lemma 4 states the drift of  $(X_t)_{t \geq 0}$  exactly [5].

**Lemma 4.**  $E(X_t - X_{t+1} \mid X_t = i) = (2 - 2^{-n+a-i+1})(1 - 1/n)^{a-i}(1/n)$  for all  $i > 0$ .

We can now supply the tail bounds, formulated as Statements (ii) and (iii) in the following theorem. The first statement is an exact expression for the expected optimization time, which has already been proved without drift analysis [5].

**Theorem 7.** *Let  $T(a)$  the time for the (1+1) EA to reach a LEADINGONES-value of at least  $a$ . Moreover, let  $r \geq 0$ . Then*

$$(i) \ E(T(a)) = \frac{n^2 - n}{2} \left( \left( 1 + \frac{1}{n-1} \right)^a - 1 \right).$$

(ii) *For  $0 < a \leq n - \log n$ , with probability at least  $1 - e^{-\Omega(rn^{-3/2})}$*

$$T(a) \leq \frac{n^2}{2} \left( \left( 1 + \frac{1}{n-1} \right)^a - 1 \right) + r.$$

(iii) *For  $\log^2 n - 1 \leq a \leq n$ , with probability at least  $1 - e^{-\Omega(rn^{-3/2})} - e^{-\Omega(\log^2 n)}$*

$$T(a) \geq \frac{n^2 - n}{2} \left( \left( 1 + \frac{1}{n-1} \right)^a - 1 - \frac{2 \log^2 n}{n-1} \right) - r.$$

### 4.2 An Application to Probabilistic Recurrence Relations

Drift analysis is not only useful in the theory of RSHs, but also in classical theoretical computer science. Here, we study a probabilistic recurrence relation of the kind  $T(n) = a(n) + T(h(n))$ , where  $n$  is the problem size,  $T(n)$  the total amount of work,  $a(n)$  the amount of work at the current level of recursion, and  $h(n)$  is a random variable, denoting the size of the problem at the next recursion level. Karp [13] studied this scenario using different probabilistic techniques than ours. Assuming knowledge of  $E(h(n))$ , he proved upper tail bounds for  $T(n)$ , more precisely he analyzed the probability of  $T(n)$  exceeding the solution of the “deterministic” process  $T(n) = a(n) + T(E(h(n)))$ .

We pick up the example from [13, Section 2.4] on the number of cycles in a permutation  $\pi \in S_n$  drawn uniformly at random, where  $S_n$  denotes the set of all permutations of the  $n$  elements  $\{1, \dots, n\}$ . A cycle is a sub-sequence of indices  $i_1, \dots, i_\ell$  such that  $\pi(i_j) = i_{(j \bmod \ell) + 1}$  for  $1 \leq j \leq \ell$ . Each permutation partitions the elements into disjoint cycles. The expected number of cycles in a random permutation is  $H_n = \ln n + \Theta(1)$ . The asymptotic probability distribution (letting  $n \rightarrow \infty$ ) of the number of cycles is well studied [1], but there are few results for finite  $n$ .

It is easy to see that the length of the cycle containing any fixed element is uniform on  $\{1, \dots, n\}$ . This gives rise to the probabilistic recurrence  $T(n) = 1 + T(h(n))$  expressing the random number of cycles, where  $h(n)$  is uniform on  $\{0, \dots, n - 1\}$ . As a result, [13] shows that the number of cycles is larger than  $\log_2(n + 1) + a$  with probability at most  $2^{-a+1}$ . Note that the  $\log_2(n)$ , which results from the solution of the deterministic recurrence, is already by a constant factor away from the expected value. Lower tail bounds are not obtained in [13]. Using our drift theorem (Theorem 2), it however follows that the number of cycles is sharply concentrated around its expectation.

**Theorem 8.** *Let  $N$  be the number of cycles in a random permutation of  $n$  elements. Then  $\Pr(N < (1 - \epsilon)(\ln n)) \leq e^{-\frac{\epsilon^2}{4}(1 - o(1)) \ln n}$  for any constant  $0 < \epsilon < 1$ . And for any constant  $\epsilon > 0$ ,  $\Pr(N \geq (1 + \epsilon)((\ln n) + 1)) \leq e^{-\frac{\min\{\epsilon, \epsilon^2\}}{6} \ln n}$ .*

For appropriate functions  $g(x)$ , our drift theorem may provide sharp concentration results for other probabilistic recurrences, such as the case  $a(n) > 1$ .

## 5 Conclusions

We have presented a new drift theorem with tail bounds. It can be understood as a general variable drift theorem and can be specialized into all existing variants of variable, additive and multiplicative drift theorems we found in the literature as well as the fitness-level technique. Moreover, it provides lower and upper tail bounds, which were not available before in the context of variable drift. Tail bounds were used to prove sharp concentration inequalities on the optimization time of the (1+1) EA on ONEMAX, linear functions and LEADINGONES. Despite the highly random fashion this RSH operates, its optimization time is highly concentrated up to lower order terms. The drift theorem also leads to tail bounds on the number of cycles in random permutations. The proofs illustrate how to use the tail bounds and we provide simplified (specialized) versions of the corresponding statements. We believe that this research helps consolidate the area of drift analysis. The general formulation of drift analysis increases our understanding of its power and limitations. The tail bounds imply more practically useful statements on the optimization time than the expected time. We expect further applications of our theorem, also to classical randomized algorithms.

**Acknowledgments.** This research received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 618091 (SAGE).

## References

1. Arratia, R., Tavaré, S.: The cycle structure of random permutations. *The Annals of Probability* **20**(3), 1567–1591 (1992)
2. Auger, A., Doerr, B., (eds.): *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific Publishing (2011)
3. Doerr, B., Fouz, M., Witt, C.: Sharp bounds by probability-generating functions and variable drift. In: *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pp. 2083–2090. ACM Press (2011)
4. Doerr, B., Goldberg, L.A.: Adaptive drift analysis. *Algorithmica* **65**(1), 224–250 (2013)
5. Doerr, B., Jansen, T., Witt, C., Zarges, C.: A method to derive fixed budget results from expected optimisation times. In: *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2013)*, pp. 1581–1588. ACM Press (2013)
6. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. *Algorithmica* **64**(4), 673–697 (2012)
7. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* **276**, 51–81 (2002)
8. Feldmann, M., Kötzing, T.: Optimizing expected path lengths with ant colony optimization using fitness proportional update. In: *Proc. of Foundations of Genetic Algorithms (FOGA 2013)*, pp. 65–74. ACM Press (2013)



9. Hajek, B.: Hitting and occupation time bounds implied by drift analysis with applications. *Advances in Applied Probability* **14**, 502–525 (1982)
10. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence. Artif. Intell.* **127**(1), 57–85 (2001), Erratum in *Artif. Intell.* **140**(1/2), 245–248 (2002)
11. Jansen, T.: *Analyzing Evolutionary Algorithms - The Computer Science Perspective*. Natural Computing Series. Springer (2013)
12. Johannsen, D.: *Random combinatorial structures and randomized search heuristics*. PhD thesis, Universität des Saarlandes, Germany (2010)
13. Karp, R.M.: Probabilistic recurrence relations. *Journal of the ACM* **41**(6), 1136–1150 (1994)
14. Lehre, P.K.: Negative drift in populations. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) *PPSN XI*. LNCS, vol. 6238, pp. 244–253. Springer, Heidelberg (2010)
15. Lehre, P.K.: Drift analysis (tutorial). In: *Companion to GECCO 2012*, pp. 1239–1258. ACM Press (2012)
16. Lehre, P.K., Witt, C.: General drift analysis with tail bounds. Technical report (2013). <http://arxiv.org/abs/1307.2559>
17. Mitavskiy, B., Rowe, J.E., Cannings, C.: Theoretical analysis of local search strategies to optimize network communication subject to preserving the total number of links. *International Journal of Intelligent Computing and Cybernetics* **2**(2), 243–284 (2009)
18. Neumann, F., Witt, C.: *Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity*. Natural Computing Series. Springer (2010)
19. Oliveto, P.S., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica* **59**(3), 369–386 (2011)
20. Oliveto, P.S., Witt, C.: Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. Technical report (2012). <http://arxiv.org/abs/1211.7184>
21. Rowe, J.E. Sudholt, D.: The choice of the offspring population size in the  $(1, \lambda)$  EA. In: *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2012)*, pp. 1349–1356. ACM Press (2012)
22. Sasak, G.H., Hajek, B.: The time complexity of maximum matching by simulated annealing. *Journal of the ACM* **35**, 387–403 (1988)
23. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Trans. on Evolutionary Computation* **17**(3), 418–435 (2013)
24. Wegener, I.: Theoretical aspects of evolutionary algorithms. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) *ICALP 2001*. LNCS, vol. 2076, pp. 64–78. Springer, Heidelberg (2001)
25. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing* **22**(2), 294–318 (2013), Preliminary version in *STACS 2012*
26. Witt, C.: Fitness levels with tail bounds for the analysis of randomized search heuristics. *Information Processing Letters* **114**(1–2), 38–41 (2014)

# **Computational Geometry III**

# Euclidean TSP with Few Inner Points in Linear Space

Paweł Gawrychowski<sup>1</sup> and Damian Rusak<sup>2</sup>

<sup>1</sup> Max-Planck-Institut für Informatik, Saarbrücken, Germany  
gawry1@gmail.com

<sup>2</sup> Institute of Computer Science, University of Wrocław, Wrocław, Poland

**Abstract.** Given a set of  $n$  points in the Euclidean plane, such that just  $k$  points are strictly inside the convex hull of the whole set, we want to find the shortest tour visiting every point. The fastest known algorithm for the version with few inner points, i.e., small  $k$ , works in  $\mathcal{O}(k^{\mathcal{O}(\sqrt{k})} k^{1.5} n^3)$  time [Knauer and Spillner, WG 2006], but also requires space of order  $k^{\mathcal{O}(\sqrt{k})} n^2$ . The best linear space algorithm takes  $\mathcal{O}(k!kn)$  time [Deineko, Hoffmann, Okamoto, Woeginger, Oper. Res. Lett. 34(1), 106-110]. We construct a linear space  $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$  time algorithm. The new insight is extending the known divide-and-conquer method based on planar separators with a matching-based argument to shrink the instance in every recursive call.

## 1 Introduction

The traveling salesman problem is one of the most natural optimization questions. It is NP-hard even in the most natural Euclidean version [6], and a simple  $\mathcal{O}(2^n n^2)$  dynamic programming can be used to solve the general version, where  $n$  is the number of points. One can do much better by exploiting the additional properties of the Euclidean variant, as independently observed by Smith [7], Kann [3], and Hwang, Chang, and Lee [2], who all applied a similar reasoning, which we will call the strategy of searching over separators, to achieve an  $\mathcal{O}(n^{\mathcal{O}(\sqrt{n})})$  running time. A natural approach to NP-hard problem is to construct an algorithm whose running time depends exponentially only on some (hopefully small) parameter  $k$  of the input. We say that a problem is *fixed-parameter tractable*, if it is possible to achieve a running time of the form  $\mathcal{O}(f(k)n^c)$ . A closely connected notion is admitting a *bikernel*, which means that we can reduce in polynomial time an instance to an instance (of a different problem) whose size is bounded by a function of  $k$ . (The notion of (bi)kernels is usually used for decision problems, while we will be working with an optimization question, but this is just a technicality.) In case of the Euclidean traveling salesman problem, a natural parameterization is to choose  $k$  to be the number of inner points, where a point is inner if it lies strictly inside the convex hull of the input. A result of Deineko, Hoffman, Okamoto and Woeginger [1] is that in such setting  $\mathcal{O}(2^k k^2 n)$

---

Supported by the *NCN* grant 2011/01/D/ST6/07164.

time can be achieved (see their paper for an explanation why such parameterization is natural). This was subsequently improved to  $\mathcal{O}(k^{\mathcal{O}(\sqrt{k})}k^{1.5}n^3)$  with a generic method given by Knauer and Spillner [4] (see the companion technical report for the details). The space usage of their method (and the previous) is superpolynomial, as they apply a dynamic programming on  $k^{\Theta(\sqrt{k})}n^2$  states.

*Contribution.* Our goal is to construct an efficient linear space algorithm. As the previously mentioned exact algorithm for the non-parametrized version [2] requires polynomial space, a natural approach is to apply the same strategy. In our case we want the total running time to depend mostly on  $k$ , though, so we devise a technique of reducing the size of current instance by applying a matching-based argument, which allows us to show that the problem admits a bikernel of quadratic size. By applying the same strategy of searching over separators on the bikernel, we achieve  $\mathcal{O}(nk^2 + k^{\mathcal{O}(k)})$  running time. To further improve on that, we extend the strategy using weighted planar separators, which give us a better handle on how the number of inner points decreases in the recursive calls. The final result is an  $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$  time linear space algorithm.

*Overview.* As in the previous papers, we start with the simple observation that the optimal traveling salesman tour visits the points on the convex hull in the cyclic order. Hence we can treat subsequent points on the convex hull as the start and end points of subpaths of the whole tour that go only through the inner points. At most  $k$  such potential subpaths include any inner points. We call them *important* and show that we can reduce (in polynomial time) the number of pairs of subsequent points on the convex hull creating the important subpaths to  $k^2$ , and for the remaining pairs we can fix the corresponding edge of the convex hull to be a part of the optimal tour, which shows that the problem admits a quadratic bikernel. The reduction shown in Section 2 is based on a matching-based argument and works in  $\mathcal{O}(nk^2 + k^6)$  time and linear space. The second step is to generalize the *Generalized Euclidean Traveling Salesman Problem* [2] as to use the properties of the convex hull more effectively. In Section 3 we modify the strategy of searching over separators, so that its running time depends mostly on the number of inner points. We use the weighted planar separator theorem of Miller [5] to prove that there exists a separator of size proportional to the square root of the number of inner points, irrespectively of the number of outer points. If the number of outer points is polynomial, which can be ensured by extending the matching-based reduction, we can iterate over all such separators. Having the separator, we guess its intersection with the solution, and recurse on the two smaller subproblems. The separator is chosen so that the number of inner points decreases by a constant factor in each subproblem, so then assuming the reduction is performed in every recursive call, we obtain the promised complexity.

*Assumptions.* We work in the Real RAM model, which ignores the issue of computing the distances only up to some accuracy.  $d(p, q)$  denotes the Euclidean distance between  $p$  and  $q$ . In the rest of the paper, by planar graph we actually

mean its fixed straight-line embedding, as the nodes will be always known points in the plane. Whenever we are talking about sets of points, we want distinct points, which can be ensured by perturbing them.

## 2 The Reduction

We want to construct an efficient algorithm for a variant of the *Euclidean Traveling Salesman Problem*, called  $k$ -ETSP, in which we are given a set  $V$  of  $n$  points such that exactly  $k$  of them lie strictly inside  $\text{CH}(V)$ , which is the convex hull of the whole set. The algorithm first reduces the problem in  $\mathcal{O}(nk^2 + k^6)$  time to an instance of the *Generalized Euclidean Traveling Salesman Problem* of size at most  $\mathcal{O}(k^2)$ , and then solves the instance in  $\mathcal{O}(k^{\mathcal{O}(\sqrt{k})})$  time. By size we mean the value of  $n + 2m$ , where  $n$  and  $m$  are defined as below.

***Generalized Euclidean Traveling Salesman Problem*  $(V, T)$ -GETSP**

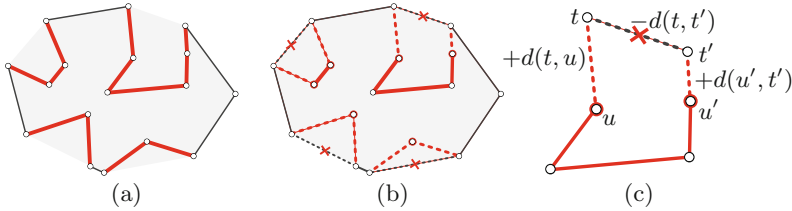
Given a set  $V = \{v_1, \dots, v_n\}$  of inner points and a set  $T = \{(t_1, t'_1), \dots, (t_m, t'_m)\}$  of terminal pairs of points, find a set of  $m$  paths with the smallest total length such that:

1. the  $i$ -th path is built on  $(t_i, t'_i)$ , i.e., starts from  $t_i$  and returns to  $t'_i$ ,
2. every  $v_i$  is included in exactly one of these paths,

assuming that in any optimal solution the paths have no self-intersections, and no path intersects other paths, except possibly at the ends.

It is well-known that in an optimal solution to an instance of  $k$ -ETSP the outer points are visited in order in which they appear on  $\text{CH}(V)$  (otherwise the solution intersects itself and can be shortened). Hence we can reduce  $k$ -ETSP to  $(V', T)$ -GETSP by setting  $V' = V \setminus \text{CH}(V)$  and  $T = \{(x_1, x_2), \dots, (x_{n-k}, x_1)\}$ , where  $\text{CH}(V) = \langle x_1, \dots, x_{n-k} \rangle$ . As any optimal solution to  $k$ -ETSP has no self-intersections, the paths in any optimal solution to the resulting instance have no self-intersections and do not intersect each other, except possibly at the ends.

We will show that given any instance of  $(V, T)$ -GETSP, we can quickly reduce the number of terminal pairs to  $\mathcal{O}(n^2)$ . A path in a solution to such instance is *important* if it includes at least one point from  $V$ , and *redundant* otherwise. Obviously, a redundant path consists of just one edge  $(t_i, t'_i)$  for some  $i$ , and the number of important paths in any solution is at most  $n$ . What is maybe less obvious, we can efficiently determine a set of at most  $n^2$  terminal pairs such that the paths built on the other terminal pairs are all redundant in some optimal solution. To prove this, we will notice that every solution to an instance of  $(V, T)$ -GETSP corresponds to a matching, and apply a simple combinatorial lemma. The idea is that every important path  $\langle u_0, u_1, \dots, u_\ell, u_{\ell+1} \rangle$  consists of the middle part  $\langle u_1, \dots, u_\ell \rangle$  containing only inner points, and the endpoints  $u_0 = t, u_{\ell+1} = t'$  for some terminal pair  $(t, t')$ . We create a weighted complete



**Fig. 1.** (a) A solution with important paths marked with thick lines, (b) connecting the inner parts of important paths to form a solution, (c) connecting a single inner part  $\{u, \dots, u'\}$  to a terminal pair  $(t, t')$  costs  $d(t, u) + d(u', t') - d(t, t')$

bipartite graph, where every possible pair of inner points  $(u, u')$  corresponds to a left vertex, and every terminal pair corresponds to a right vertex. The weight of an edge between  $(u, u')$  and  $(t, t')$  is  $d(t, u) + d(u', t') - d(t, t')$ , see Fig. 1(c).

First we present a simple combinatorial lemma. Given a weighted complete bipartite graph  $G = (U \cup V, U \times V, c)$ , where  $c(u, v)$  is the weight of an edge  $(u, v)$ ,  $\text{cost}(X, Y)$  denotes the weight of a cheapest matching of  $X \subseteq U$  to  $Y \subseteq V$ , if  $|X| \leq |Y|$ . If  $M$  is a matching of  $X$  to  $Y$ , then we denote by  $M[X]$  and  $M[Y]$  the subsets of  $X$  and  $Y$  matched by  $M$ .

**Lemma 1.** *Let  $G = (U \cup V, U \times V, c)$  be a weighted complete bipartite graph, where  $|U| \leq |V|$ . If  $M_{\min}$  is a cheapest matching of  $U$  to  $V$ , then for every  $A \subseteq U$  we have  $\text{cost}(A, M_{\min}[V]) = \text{cost}(A, V)$ .*

*Proof.* Assume the opposite, i.e., there is some  $A \subseteq U$  such that for any cheapest matching  $M$  of  $A$  to  $V$  we have  $M[V] \not\subseteq M_{\min}[V]$ . Fix such  $A$  and take any cheapest matching  $M$  of  $A$  to  $V$ . If there are multiple such  $M$ , take the one with the largest  $|M[V] \cap M_{\min}[V]|$ . Then look at  $M \oplus M_{\min}$ , which is the set of edges belonging to exactly one of  $M$  and  $M_{\min}$ . It consists of node-disjoint alternating cycles and alternating paths, and the alternating paths can be of either odd or even length. Because  $M[V] \not\subseteq M_{\min}[V]$ , there is a vertex  $x \in V$  such that  $x \in M[V] \setminus M_{\min}[V]$ . It is clear that there is a (nontrivial) path  $P$  starting at  $x$ , as  $x \in M[V]$  but  $x \notin M_{\min}[V]$ . We want argue that its length is even. Its first edge comes from  $M$ , so if the total length is odd, then the last edge comes from  $M$  as well, so the path ends at a vertex  $y \in A$ . But all such  $y$  are matched in  $M_{\min}$ , so  $P$  cannot end there. Hence it ends at a vertex  $y \in M_{\min}[V] \setminus M[V]$ , and its length is even. Now we consider three cases depending on the sign of  $\text{cost}(P)$ , which is the total weight of all edges in  $P \cap M_{\min}$  minus the total weight of all edges in  $P \cap M$ :

1. if  $\text{cost}(P) > 0$  then  $M_{\min} \oplus P$  is cheaper than  $M_{\min}$ , so  $M_{\min}$  was not a cheapest matching of  $U$  to  $V$ ,
2. if  $\text{cost}(P) < 0$  then  $M \oplus P$  is cheaper than  $M$ , so  $M$  was not a cheapest matching of  $A$  to  $V$ ,

3. if  $\text{cost}(P) = 0$ , then  $M' = M \oplus P$  is a cheapest matching of  $A$  to  $V$ , and  $|M'[V] \cap M_{\min}[V]| > |M[V] \cap M_{\min}[V]|$ , so  $M$  was not a cheapest matching of  $A$  to  $V$  with the largest  $|M[V] \cap M_{\min}[V]|$  in case of a tie.

Hence there is a cheapest matching  $M$  of  $A$  to  $V$  such that  $M[V] \subseteq M_{\min}[V]$ .  $\square$

**Lemma 2.** *With a read-only constant-time access to a weighted complete bipartite graph  $G = (U \cup V, U \times V, c)$ , where  $|U| \leq |V|$ , we can find a cheapest matching of  $U$  to  $V$  in  $\mathcal{O}(|U|^3 + |U||V|)$  time and  $\mathcal{O}(|V|)$  space.*

*Proof.* Let  $p = |U|$  and  $q = |V|$ . The naive approach would be to find a cheapest matching using  $p$  iterations of Dijkstra’s algorithm implemented with a Fibonacci heap in  $\mathcal{O}(p + q)$  space and  $\mathcal{O}(p((p + q) \log(p + q) + pq))$  total time. The claimed complexities can be achieved by appropriately truncating the input graph.  $\square$

**Theorem 1.** *Given an instance of  $(V, T)$ -GETSP with  $m \geq n^2$ , we can find  $T_0 \subseteq T$  of size  $m - n^2$ , such that there is an optimal solution in which the paths built on pairs from  $T_0$  are all redundant, in  $\mathcal{O}(mn^2 + n^6)$  time and  $\mathcal{O}(m)$  space.*

*Proof.* Let  $W = V \times V$  and  $G = (W \cup T, W \times T, c)$  be a weighted complete bipartite graph with  $W$  and  $T$  as the left and right vertices, respectively. The weight of an edge connecting  $(v, v')$  and  $(t, t')$  is defined as  $c((v, v'), (t, t')) = d(t, v) + d(v', t') - d(t, t')$ . Informally, given a path  $\langle v, \dots, v' \rangle$  consisting of inner points,  $c((v, v'), (t, t'))$  is the cost of replacing a redundant path  $\langle t, t' \rangle$  with an important path  $\langle t, v, \dots, v', t' \rangle$ , assuming that we have already taken into account the length of the inner part  $\langle v, \dots, v' \rangle$ , see Fig. 1(c). Now any solution corresponds to a matching in  $G$ , because for every important path  $\langle t_i, v_i, \dots, v'_i, t'_i \rangle$  we can match  $(v_i, v'_i)$  to  $(t_i, t'_i)$ . More precisely, if we denote by  $i_1 < \dots < i_s$  the indices of all these important paths and fix their inner parts  $\langle v_{i_j}, \dots, v'_{i_j} \rangle$ , then the solution corresponds to a matching of  $W' = \{(v_{i_1}, v'_{i_1}), \dots, (v_{i_s}, v'_{i_s})\}$  to  $T$ , and the cost of the solution is equal to the total length of all inner parts plus  $\sum_i d(t_i, t'_i)$  plus the cost of the matching. In the other direction, any matching of  $W'$  to  $T$  corresponds to a solution with the given set of inner parts (but possibly different indices of important paths). The cost of that solution is, again, equal to the total length of all inner parts plus  $\sum_i d(t_i, t'_i)$  plus the cost of the matching, so any cheapest matching corresponds to an optimal solution. By Lemma 1 we know, that  $\text{cost}(W', M_{\min}[T]) = \text{cost}(W', T)$ , so there is always a cheapest matching of  $W'$  to  $T$  which uses only the nodes in  $M_{\min}[T]$ , where  $M_{\min}$  is a cheapest matching of  $W$  to  $T$  in the whole  $G$ . Therefore, we can set  $T_0 = T \setminus M_{\min}[T]$ , because there is at least one optimal solution, where the paths built on pairs from such  $T_0$  are all redundant. Clearly,  $|T_0| = m - n^2$ . Finally, Lemma 2 allows us to construct a cheapest matching efficiently, as we can implement read-only access to any  $c((v, v'), (t, t'))$  without explicitly storing the graph, so the total space usage is  $\mathcal{O}(n)$  and the total time complexity is  $\mathcal{O}(mn^2 + n^6)$  as claimed.  $\square$

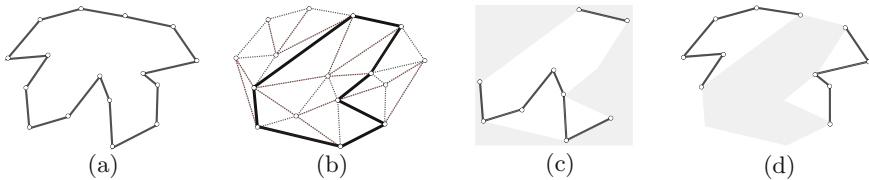
### 3 Searching over Separators

In this section we briefly recap the method of searching over separators used in [2] to solve the *Euclidean Traveling Salesman Problem*. At a high level, it is a divide-and-conquer algorithm. We know that an optimal solution has no self-intersections, hence we can treat it as a planar graph. Every planar graph has a small simple cycle separator, which is a simple cycle, which can be removed as to split the whole graph into smaller pieces. Such separator can be used to divide the original problem into smaller subproblems, which are then solved recursively.

**Theorem 2 (Miller [5]).** *In any 2-connected planar graph with nonnegative weights summing up to 1 assigned to nodes, there exists a simple cycle, called a simple cycle separator, on at most  $2\sqrt{2} \lfloor d/2 \rfloor \sqrt{N}$  vertices, dividing the graph into the interior and the exterior part, such that the sum of the weights in each part is at most  $\frac{2}{3}$ , where  $d$  is the maximum face size and  $N$  is the number of nodes.*

Consider an instance of  $(V, T)$ -GETSP and its optimal solution, which by the assumption has no self-intersections, so there exists a planar graph such that any edge of the solution appears there. Then by Theorem 2 there is a simple cycle on at most  $2\sqrt{2} \lfloor d/2 \rfloor (n + 2m)$  nodes such that any edge of the solution is either completely outside, completely inside, or lies on the cycle, and furthermore there are at most  $\frac{2}{3}(n + 2m)$  points in either the exterior and the interior part. We want the cycle to be small, so we need to bound  $d$ . For all inner faces, this can be ensured by simply triangulating them. To ensure that the outer face is small, we add three enclosing points.

Now consider how the paths in the solution intersect with the simple cycle separator. Each path is either completely outside, completely inside, or intersects with one of the nodes of the separator. Any such intersecting path can be partitioned into shorter subpaths, such that the endpoints of the subpaths are either the endpoints of the original paths or the nodes of the separator, and every subpath is outside or inside, meaning that all of its inner nodes are completely outside or completely inside. This suggests that we can create two smaller instances of  $(V, T)$ -GETSP corresponding to the interior and the exterior part of the graph, such that the solutions of these two smaller subproblems can be merged to create the solution for the original problem, see Fig. 2.



**Fig. 2.** (a) A solution, (b) the triangulated planar graph and its simple cycle separator, (c) a solution to the interior subproblem, (d) a solution to the exterior problem



Of course we don't know the solution, so we cannot really find a simple cycle separator in its corresponding triangulated planar graph. But the size of the separator is at most  $c\sqrt{n+2m+3}$  for some constant  $c$ , so we can iterate over all possible simple cycles of such length, and for every such cycle check if it partitions the instance into two parts of sufficiently small sizes. The number of cycles is at most  $c\sqrt{n+2m+3} \binom{n+2m+3}{c\sqrt{n+2m+3}} (c\sqrt{n+2m+3})!$ , which is  $\mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})})$ .

Similarly, because we don't know the solution, we cannot check how it intersects with our simple cycle separator, but we can iterate over all possibilities. To bound the number of possibilities, we must be a little bit more precise about what intersection with the separator means. We create a number of new terminal pairs. Every node of the separator appears in one or two of these new terminal pairs. The new terminal pairs might contain some of the original terminal points, under the restriction that for any original terminal pair, either none of its points are used in the new terminal pairs, or both are (and in the latter case, we remove the original terminal pair), and no sequence of new terminal pairs creates a cycle, i.e., there is no  $(p_1, p_2), \dots, (p_\ell, p_1)$  with  $\ell \geq 3$ . Then, for every new terminal pair  $(p, p')$ , we decide if its path lies fully within the exterior or the interior part (if it directly connects two consecutive points on the cycle, we can consider it as belonging to either part). If  $p$  is one of the original terminal points, and  $p'$  is a new terminal point, then the corresponding path lies fully within the part where  $p$  belongs to. Such a choice allows us to partition the original problem into two smaller subproblems, so that their optimal solutions can be merged to recover the whole solution, and that the subproblems are smaller instances of  $(V, T)$ -GETSP. Hence iterating over all choices and choosing an optimal solution in every subproblem allows us to solve the original instance.

To bound the number of choices, the whole process can be seen as partitioning the nodes of the separator into ordered subsets, selecting two of the original terminal points for every of these subsets, and finally guessing, for every two nodes subsequent in one of the subsets, whether the path connecting them belongs to the exterior or the interior part. We must also check if for any original pair  $(p, p')$  either none of its points was selected, or both of them were, but even without this last easy check the number of possibilities is bounded by  $B_{c\sqrt{n+2m+3}} (c\sqrt{n+2m+3})! \binom{2m}{2c\sqrt{n+2m+3}} 2^{c\sqrt{n+2m+3}}$ , where  $B_s$  is the  $s$ -th Bell number. This is, again,  $\mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})})$ .

The algorithm iterates over all separators and over all possibilities of how the solution intersects with each of them. For each choice, it recurses on the resulting two smaller subproblems, and combines their solutions. Even though we cannot guarantee that all optimal solutions in these subproblems have no self-intersections, any optimal solution to the original problem has such property, so for at least one choice the subproblems will have such property, which is enough for the correctness. Because the size of every subproblem is at most  $b = \frac{2}{3}(n+2m) + c\sqrt{n+2m+3}$ , the recurrence for the total running time is:  $T(n+2m) = \mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})}) \cdot 2T(b)$ . For large enough  $n+2m$ , we have that  $b \leq \frac{3}{4}(n+2m)$ , and the recurrence solves to  $T(n+2m) = \mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})})$ . The

space complexity is linear, because we only need to generate the subproblems, which requires iterating over all subsets and all partitions into ordered subsets.

By first applying Theorem 1 and then using the method of searching over separators on the resulting graph, we can solve an instance of  $k$ -ETSP in  $\mathcal{O}(nk^2 + k^{\mathcal{O}(k)})$  time and linear space.

### 4 $(V, T, H)$ -GETSP

To extend the divide-and-conquer algorithm described in the previous section, we need to work with a version of  $(V, T)$ -GETSP, which is more sensitive to the number of terminal pairs such that both points belong to the convex hull. We call the extended version  $(V, T, H)$ -GETSP, and define its size to be  $n + 2m + 2\ell$ . Given an instance of  $k$ -ETSP, we can reduce the problem to solving an instance of  $(V, T, H)$ -GETSP with  $|V| = k$ ,  $|T| = 0$ , and  $|H| = n - k$ .

**Generalized Euclidean Traveling Salesman Problem  $(V, T, H)$ -GETSP**

Given a set  $V = \{v_1, \dots, v_n\}$  of inner points, a set  $T = \{(t_1, t'_1), \dots, (t_m, t'_m)\}$  of terminal pairs of points, and a set  $H = \{(h_1, h'_1), \dots, (h_\ell, h'_\ell)\}$  of hull pairs of points, where for any  $i$  the point  $h_i$  and  $h'_i$  are neighbors on the convex hull of the set of all points<sup>1</sup>, find a set of  $m + \ell$  paths with the smallest total length such that:

1. the  $i$ -th path is built on  $(t_i, t'_i)$ , for  $i = 1, 2, \dots, m$ ,
2. the  $m + i$ -th path is built on  $(h_i, h'_i)$ , for  $i = 1, 2, \dots, \ell$ ,
3. every  $v_i$  is included in exactly one of these paths,

assuming that in any optimal solution the paths have no self-intersections, and no path intersects other paths, except possibly at the ends.

We will show that if  $\ell = \text{poly}(n)$ , then  $(V, T, H)$ -GETSP can be solved in  $\mathcal{O}((n + 2m)^{\mathcal{O}(\sqrt{n+2m})})$  time and linear space using an extension of the method from the previous section. Combined with Theorem 1, this gives an  $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$  time and linear space solution for  $k$ -ETSP. First we extend Theorem 1.

**Lemma 3.** *Given an instance of  $(V, T, H)$ -GETSP with  $n = |V|$ ,  $m = |T|$ , and  $\ell = |H|$ , where  $m + \ell \geq n^2$ , we can find  $T_0 \subseteq T$  and  $H_0 \subseteq H$  such that  $|T_0| + |H_0| = m + \ell - n^2$ , such that there is an optimal solution in which the paths built on pairs from  $T_0 \cup H_0$  are all redundant in  $\mathcal{O}((m + \ell)n^2 + n^6)$  time and  $\mathcal{O}(m + \ell)$  space.*

Recall that the recursive method described in the previous section iterates over simple cycle separators. Because now the (unknown) graph is on  $n + 2m + 2\ell$  vertices, the best bound on the length of the separator that we could directly get from Theorem 2 is  $c\sqrt{n + 2m + 2\ell + 3}$ , which is too large. But say that we

<sup>1</sup> Other points given in the input might or might not lie on the convex hull.

could show that there exists a simple cycle separator of length  $\mathcal{O}(\sqrt{n + 2m + 2})$ , such that the value of  $n + 2m$  decreases by a constant factor in both parts. Iterating over all such simple cycle separators takes  $\mathcal{O}((n + 2m + 2)^\ell)^{\mathcal{O}(\sqrt{n + 2m})}$  time, and iterating over all possibilities of how the separator intersects with the solution then takes  $B_{\mathcal{O}(\sqrt{n + 2m})} \mathcal{O}(\sqrt{n + 2m})! \binom{n + 2m + 2\ell}{\mathcal{O}(\sqrt{n + 2m})} 2^{\mathcal{O}(\sqrt{n + 2m})}$  time. All in all, the total number of possibilities becomes  $\mathcal{O}((n + 2m + 2\ell)^{\mathcal{O}(\sqrt{n + 2m})})$ , which assuming that  $\ell = \text{poly}(n)$  is  $\mathcal{O}((n + 2m)^{\mathcal{O}(\sqrt{n + 2m})})$ . Applying this reasoning in a recursive manner as in the previous section results in Algorithm 1. Compared to the algorithm from the previous section, the changes are:

1. in every recursive call we reduce the number of terminal and hull pairs using Lemma 3,
2. we add just two enclosing points (instead of three) as explained in Lemma 4,
3. when forming the subproblems, we might connect both some terminal points and some hull points with the nodes of the separator, and in the latter case, the new pair always becomes a terminal pair in the subproblem.

If  $\ell = \text{poly}(n)$  in the original instance, then we can maintain such invariant in all recursive calls without increasing the running time, because the (polynomial) cost of the reduction in a subproblem can be charged to its parent. Therefore, because the value of  $n + 2m$  decreases by a constant factor in both subproblems, the total time is  $\mathcal{O}((n + 2m)^{\mathcal{O}(\sqrt{n + 2m})})$  by the same recurrence as previously.

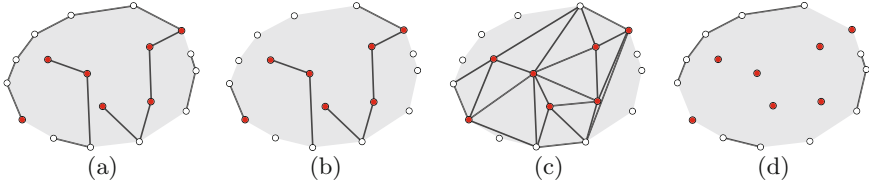
Now the goal is to prove that it is enough to consider simple cycle separators of length  $c\sqrt{n + 2m + 2}$ . To this end, we will prove that there exists a planar graph with the following properties:

- (a) its set of nodes includes all inner and terminal points together with the two enclosing points, and possibly some hull points,
- (b) any edge from the solution is either its edge, or lies within one of its faces,
- (c) all of its faces are of size at most 4 and its size is  $\mathcal{O}(n + 2m)$ .

If such a graph exists, then by Theorem 2 it has a simple cycle separator of size  $\mathcal{O}(\sqrt{n + 2m})$  due to (c). By assigning equal weights summing up to one to all inner and terminal points, which by (a) are nodes of the graph, and zero weights to the remaining nodes, we get a simple cycle separator which, by (b), divides the original problem into two subproblems, such that the optimal solution to the subproblems can be combined to form an optimal solution to the original problem, and there are at most  $\frac{2}{3}(n + 2m)$  inner and terminal points in every subproblem, so Algorithm 1 is correct. Before we show that such a graph exists, we explain how to choose the enclosing points.

**Lemma 4.** *For any set of points  $A$  we can find two enclosing points  $I_1, I_2$ , lying outside  $\text{CH}(A)$ , and two nodes of  $\text{CH}(A)$  called  $v_{up}, v_{down}$ , such that:*

1. all points of  $\text{CH}(A)$  between  $v_{up}$  and  $v_{down}$  (clockwise) lie inside  $\triangle I_2 v_{up} v_{down}$ , and all points of  $\text{CH}(A)$  between  $v_{down}$  and  $v_{up}$  lie inside  $\triangle I_1 v_{up} v_{down}$ ,
2. for any point  $w$  of  $\text{CH}(A)$  between  $v_{up}$  and  $v_{down}$ ,  $I_1 w$  has no common point with  $\text{CH}(A)$  except for  $w$ , and for any  $w$  between  $v_{down}$  and  $v_{up}$ ,  $I_2 w$  has no common point with  $\text{CH}(A)$  except for  $w$ .



**Fig. 3.** (a) The segments in  $S$ , (b) the segments in  $S'$ , (c) the initial triangulated planar graph, (d) the remaining segments. Points from  $U$  are filled.

Any optimal solution to an instance of  $(V, T, H)$ -GETSP corresponds to a *hull structure*  $(U, H, S)$ , where  $U$  contains all inner and terminal points,  $H$  contains all hull points, and  $S$  contains all segments constituting the paths, meaning that:

1.  $U$  and  $H$  are two sets of points in the plane with  $H \subseteq \text{CH}(U \cup H)$ ,
2.  $S$  is a collection of segments connecting the points in  $U \cup H$  such that no segment intersects other segment, except possibly at the ends,
3. any point from  $U \cup H$  is an endpoint of at most two segments in  $S$ ,
4. every segment in  $S$  connecting two points from  $H$  lies on  $\text{CH}(U \cup H)$ .

**Lemma 5.** *If  $(U, H, S)$  is a hull structure, and  $I_1, I_2$  are the points enclosing  $U \cup H$ , then there exists a planar graph, such that:*

1. the nodes are all points from  $U \cup \{I_1, I_2\}$  and possibly some points from  $H$ ,
2. any segment from  $S$  is either an edge of the graph, or lies within its face,
3. all of its faces are of size at most 4,
4. the size of the graph is  $\mathcal{O}(|U|)$ .

*Proof.* The enclosing points  $I_1, I_2$  are defined by applying Lemma 4 on  $U \cup H$ , same for  $v_{up}$  and  $v_{down}$ . The subsets of  $U \cup H$  on the same side of the line going through  $v_{up}v_{down}$  as  $I_1$  and  $I_2$  will be called  $V_1$  and  $V_2$ , respectively. The subset of  $S$  containing all segments with at least one endpoint in  $U$  will be called  $S'$ . Because any point of  $U$  is an endpoint of at most two segments,  $|S'| = \mathcal{O}(|U|)$ . We define  $U'$  to be the whole  $U$  together with the points of  $H$  which are an endpoint of some segment in  $S'$ , and create the first approximation of the desired planar graph using  $U'$  as its set of nodes, and  $S'$  as its set of edges. We triangulate this planar graph, so that its inner faces are of size 3. Notice that all of its edges are inside or on  $\text{CH}(U')$ , and all remaining segments in  $S \setminus S'$  lie on  $\text{CH}(U \cup H)$ , see Fig. 3. So far, the size of the planar graph is  $\mathcal{O}(|U|)$ , its faces are small, and the nodes are all points from  $U$  and possible some points from  $H$ , and any segment from  $S'$  is an edge there. Therefore, we just need to make sure that any remaining segment is either an edge, or lies within a face.

To deal with the remaining segments, we add  $I_1$  and  $I_2$  to the set of nodes. Fix any point  $P$  strictly inside  $\text{CH}(U')$  and, for every node  $P'$  of  $\text{CH}(U')$ , draw a ray starting in  $P$  and going through  $P'$ . All these rays partition the region outside  $\text{CH}(U')$  into convex subregions  $R_1, R_2, \dots, R_{|\text{CH}(U')|}$ . The intersection

of  $R_i$  with  $\text{CH}(U \cup H)$ , called  $T_i$ , contains exactly two vertices of  $\text{CH}(U')$ , call them  $y_i$  and  $y'_i$ , see Fig 4(a). We will process every such  $T_i$  separately, extending the current graph by adding new triangles. Consider the sequence of points  $v_{a_i}, v_{a_i+1}, \dots, v_{b_i}$  of  $\text{CH}(U \cup H)$ , which belong to  $T_i$ . If the sequence is empty, there is nothing to do. Otherwise we have two cases:

1. if  $a_i = b_i$ , create a new triangle  $\Delta v_{b_i} y_i y'_i$  to  $\mathcal{T}$ , see Fig. 4(b),
2. if  $a_i \neq b_i$ , create two new triangles  $\Delta v_{a_i} y_i y'_i$ ,  $\Delta v_{a_i} v_{b_i} y'_i$ . Then add a triangle  $\Delta I_j v_{a_i} v_{b_i}$  if both  $v_{a_i}$  and  $v_{b_i}$  belong to the same  $V_j$ , see Fig. 5(a). Otherwise either  $v_{up}$  or  $v_{down}$  is in  $v_{a_i}, \dots, v_{b_i}$ , and we add three triangles as in Fig. 5(b).

Now any remaining segment which lies within a single  $T_i$  is inside one of the new triangles. For each such segment  $v_j v_{j+1}$  we simply add either  $\Delta I_1 v_j v_{j+1}$  or  $\Delta I_2 v_j v_{j+1}$ , see Fig. 5(c). This is correct because any two consecutive points on  $\text{CH}(H)$  always either both belong to  $V_1$  or both belong to  $V_2$ . One ray can cross at most one edge, so the number of triangles created in this step is at most  $|U'|$ .

By the construction, the insides of any new triangles are disjoint. Also, they all lie outside  $\text{CH}(U')$ . Hence we can add the new triangles to the initial planar graph to form a larger planar graph. Because we created  $\mathcal{O}(|U'|)$  new triangles, the size of the new planar graph is still  $\mathcal{O}(|U|)$ , though. Now some of its faces might be large, though, so we include  $I_1, I_2, v_{up}, v_{down}$  in its set of nodes, and all  $I_1 v_{up}, I_1 v_{down}, I_2 v_{up}, I_2 v_{down}$  in its set of edges. Finally, we triangulate the large inner faces, if any. The size of the final graph is  $\mathcal{O}(|U|)$  and we ensured that any segment from  $S$  is either among its edges, or lies within one of its faces.  $\square$

Lemma 5 shows that it is indeed enough to iterate over separators of size  $c\sqrt{n + 2m + 2}$ , hence Algorithm 1 is correct. The remaining part is to argue that it needs just linear space. By Lemma 3, the reduction uses  $\mathcal{O}(m + \ell)$  additional space which can be immediately reused. Iterating through all ordered subsets of size at most  $c\sqrt{n + 2m + 2}$  can be easily done with  $\mathcal{O}(\sqrt{n + 2m})$  additional space. Bounding the space necessary to iterate over all possibilities of how the solution intersects with the separator is less obvious, but the same bound can be derived by looking at how the possibilities were counted. The  $\mathcal{O}(\sqrt{n + 2m})$  additional space must be stored for every recursive call. Additionally, for each call we must store its arguments  $V, T$  and  $H$ , which takes  $\mathcal{O}(n + 2m + 2\ell)$

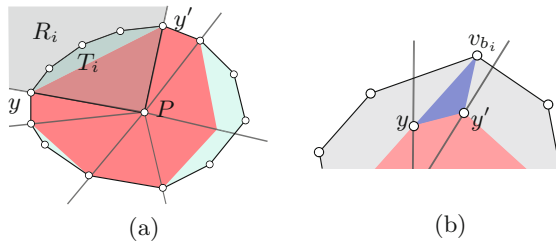
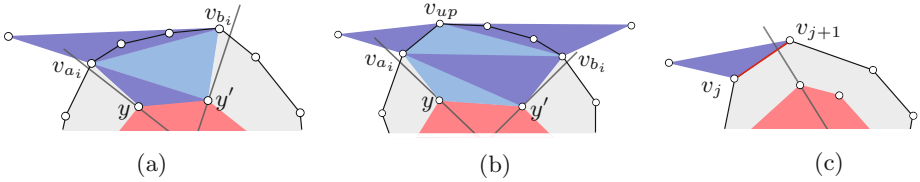


Fig. 4. (a) The intersection  $T_i$ , (b) adding one triangle when  $a_i = b_i$



**Fig. 5.** (a)  $v_{a_i}, v_{b_i}$  in the same  $V_j$ , (b)  $v_{up}$  between  $v_{a_i}$  and  $v_{b_i}$ , (c) a remaining segment

additional space. As  $n + 2m$  decreases by a constant factor in every recursive call, the recursion depth is  $\mathcal{O}(\log(n + 2m))$ , which in turn implies  $\mathcal{O}(n + 2m + 2\ell \log(n + 2m))$  overall space consumption. Even though we always reduce the instance so that  $\ell \leq n^2$ , this bound might be superlinear.

Recall that the hull pairs in the subproblems are disjoint subsets of all hull pairs in the original problem. Hence, instead of copying the hull pairs into the subproblems, we can store them in one global array. Before the recursive calls, we rearrange the fragment so that the hull pairs which should be processed in both subproblems are, stored in contiguous fragments of the global array. The rearranging can be done in linear space and constant additional space. The remaining issue is that when we return from the subproblems, the fragment containing the hull pairs might have been arbitrarily shuffled. This is a problem, because we are iterating over the ordered subsets of all points, which requires operating on their indices. Now the order of the hull pairs might change, so we cannot identify a hull point by storing the index of its pair. Nevertheless, we can maintain an invariant that all hull pairs in the current problem are lexicographically sorted. In the very beginning, we just sort the global array. Then, before we recurse on a subproblem, we make sure that its fragment is sorted. After we are done with both subproblems, we re-sort the fragment of the global array corresponding to the current problem. This decreases the overall space complexity to linear.

**Theorem 3.**  *$k$ -ETSP can be solved in  $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$  time and linear space.*

## References

1. Deineko, V.G., Hoffmann, M., Okamoto, Y., Woeginger, G.J.: The traveling salesman problem with few inner points. *Operations Research Letters* **34**(1), 106–110 (2006)
2. Hwang, R., Chang, R., Lee, R.: The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica* **9**(4), 398–423 (1993)
3. Kann, V.: *On the Approximability of NP-complete Optimization Problems*. Trita-NA, Royal Institute of Technology, Department of Numerical Analysis and Computing Science (1992)
4. Knauer, C., Spillner, A.: A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In: Fomin, F.V. (ed.) *WG 2006*. LNCS, vol. 4271, pp. 49–57. Springer, Heidelberg (2006)

5. Miller, G.L.: Finding small simple cycle separators for 2-connected planar graphs. In: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, STOC 1984, pp. 376–382. ACM, New York (1984)
6. Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* **4**(3), 237–244 (1977)
7. Smith, W.D.: *Studies in Computational Geometry Motivated by Mesh Generation*. Ph.D. thesis, Princeton University, Princeton, NJ, USA (1989)

# Bottleneck Partial-Matching Voronoi Diagrams and Applications

Matthias Henze<sup>(✉)</sup> and Rafel Jaume<sup>(✉)</sup>

Institut für Informatik, Freie Universität Berlin,  
Takustraße 9, 14195 Berlin, Germany  
matthias.henze@fu-berlin.de, jaume@mi.fu-berlin.de

**Abstract.** Given two finite point sets  $A$  and  $B$  in the plane, we study the minimization of the bottleneck distance between the smaller set  $B$  and an equally-sized subset of  $A$  under translations. We relate this problem to a Voronoi-type diagram and derive polynomial bounds for its complexity that are tight in the size of  $A$ . We devise efficient algorithms for the construction of such a diagram and its lexicographic variant, which generalize to higher dimensions. We use the diagram to find the optimal bottleneck matching under translations, and to compute a connecting path of minimum bottleneck cost between two positions of  $B$ .

## 1 Introduction

Applications often demand algorithms to find an occurrence of a point pattern in a bigger set of points. It is also common to define a similarity measure between the pattern and the point set as the minimum among the images of the pattern under a set of allowed transformations.

One of the most studied similarity measures for two finite point sets  $A$  and  $B$  in  $\mathbb{R}^d$  is the *directed Hausdorff distance*, which is the maximum of the (Euclidean) distances from each point in  $B$  to its nearest neighbor in  $A$ . For some applications, it is required that each point of the smaller set is matched to a distinct point in the bigger one. The resulting distance is called the *bottleneck distance* and was introduced for equally-sized sets in [2] as

$$\Delta(B, A) = \min_{\sigma: B \rightarrow A} \max_{b \in B} \|b - \sigma(b)\|,$$

where  $\|\cdot\|$  denotes the Euclidean norm and the minimum is taken over all injections from  $B$  into  $A$ . In contrast to the directed Hausdorff distance, the bottleneck distance has the advantage of being symmetric for equally-sized sets. On the other hand, it is harder to compute, since the points cannot be regarded independently. Note that there might be several matchings that minimize the bottleneck distance, even when all the distances between points are distinct.

---

Research by the first author is supported by the ESF EUROCORES programme EuroGIGA-VORONOI, (DFG): RO 2338/5-1, and by the second author by DAAD.



When this is to be avoided, the matching that lexicographically minimizes the distances between matched points is often considered [6, 10, 18].

In this paper, we are interested in a dynamic version of the bottleneck distance. More precisely, we want to efficiently compute the minimum bottleneck distance among all translated copies of  $B$  with respect to  $A$ , that is,  $\min_{t \in \mathbb{R}^d} \Delta(B + t, A)$ . This problem will be called *bottleneck partial-matching under translations*. It was introduced for equally-sized point sets in the plane by Alt et al. [2], who gave an algorithm running in  $O(n^6 \log n)$  time for point sets of size  $n$ . Their bound was improved to  $O(n^5 \log^2 n)$  by Efrat et al. [12].

To the best of our knowledge, bottleneck matching under translations has not been studied with the focus on algorithms whose complexity is sensitive to the size of the smaller set. In order to do so, we associate Voronoi-type diagrams to the problem, which we call *bottleneck diagrams* and *lex-bottleneck diagrams*, respectively. This follows an idea of Rote [17] who partitioned the space of translations according to the (partial) matching that minimizes the least-squares distance between translated copies of  $B$  and  $A$  (cf. [3, 15] for follow-up studies). Our diagrams partition  $\mathbb{R}^d$  into polyhedral cells that correspond to locally-optimal (lexicographic-)bottleneck matchings. A non-archival abstract containing part of our studies appeared in [14].

In Section 2, we formally introduce the Voronoi-type diagrams before investigating their basic properties and combinatorial complexity. It turns out that there exists a (lex-)bottleneck diagram of complexity  $O(n^2 k^6)$  for any given planar point sets  $A, B$  with  $k = |B| \leq |A| = n$ , and that this bound cannot be improved with respect to the size of  $A$ . Based on these complexity results, we devise construction algorithms in Section 3 that run in time  $O(n^2 k^8)$  for bottleneck diagrams and in time  $O(n^2 k^{10})$  for lex-bottleneck diagrams. Once having constructed a bottleneck diagram, the partial-matching problem under translations can be solved in the same asymptotic running time. A  $k$ -sensitive analysis of the algorithm of Alt et al. shows that it solves the bottleneck partial-matching problem in  $O(n^3 k^3 \log n)$  time, which implies that our method outperforms it whenever  $k = O((n \log n)^{1/5})$ . Moreover, our study of bottleneck diagrams generalizes easily to high dimensions and has applications toward related questions. For instance, as shown in Section 4, we can use the bottleneck diagram to compute a *bottleneck path* between two given positions of a pattern in the plane.

## 2 Bottleneck Partial-Matching Voronoi Diagrams

In this section, we introduce the bottleneck partial-matching Voronoi diagram and its lexicographic variant. We identify basic properties of these diagrams which help us to establish bounds on their combinatorial complexity.

Throughout the paper, we assume that we are given two point sets  $A, B \subset \mathbb{R}^d$  with  $k = |B| \leq |A| = n$  and that  $B$  is allowed to be translated. We use the term *edge* for a pair of points  $(a, b) \in A \times B$  and denote it by  $ab$  for short. The *length* of the edge  $ab$  is defined as the Euclidean distance  $\|b - a\|$ . In this context, we will identify every injection of  $B$  into  $A$  with the *matching* (set of edges) it

induces. Our aim is to subdivide the space of translations in a way that, for every translation of  $B$ , we can retrieve a matching whose longest edge is as short as possible. For the ease of presentation, we assume that there are no two different pairs  $(a, b), (a', b') \in A \times B$  such that  $a - b = a' - b'$ . However, it can be shown that the bounds on the complexity and running time in this paper still hold if the point sets violate this general-position assumption. Given  $m \in \mathbb{N}$ , we denote by  $[m]$  the set  $\{1, \dots, m\}$ .

**Definition 1.** A bottleneck matching for two finite point sets  $A, B \subset \mathbb{R}^d$  with  $|B| \leq |A|$  is an injection  $\pi : B \hookrightarrow A$  such that

$$\max_{b \in B} \|b - \pi(b)\| \leq \max_{b \in B} \|b - \sigma(b)\|,$$

for every other matching  $\sigma$ . The bottleneck cost of a matching  $\sigma$  is the function  $f_\sigma : \mathbb{R}^d \rightarrow \mathbb{R}$  defined as

$$f_\sigma(t) = \max_{b \in B} \|b + t - \sigma(b)\|^2.$$

The bottleneck value of a translation  $t \in \mathbb{R}^d$  is

$$\mathcal{E}(t) = \min_{\sigma: B \hookrightarrow A} f_\sigma(t).$$

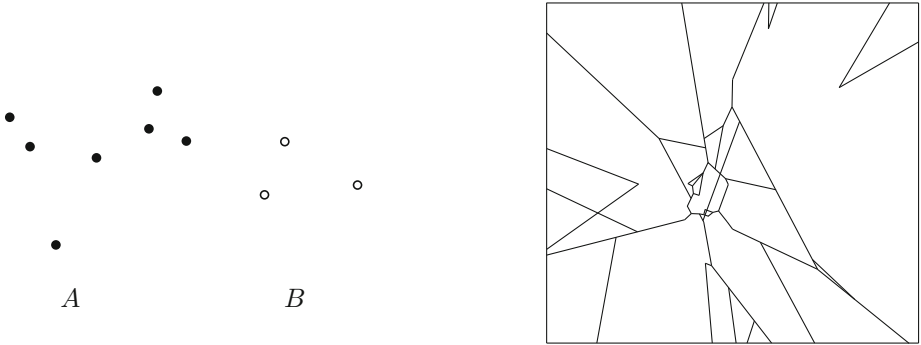
Note that whether a matching is bottleneck depends on the Euclidean length of its edges while the cost of a matching depends on the square of it. This squaring does not change our problem and will be convenient later on.

For a given translation  $t \in \mathbb{R}^d$ , a bottleneck matching for  $B + t$  and  $A$ , and hence the value  $\mathcal{E}(t)$ , can be computed applying the algorithms described at the beginning of Section 3. However, we are interested in finding a bottleneck matching for every fixed position of the point set  $B$ .

Observe that there is no one-to-one correspondence between quadratic pieces of  $\mathcal{E}$  and bottleneck matchings. In fact, it is not hard to see that for an open set of positions of  $B$ , there may be  $\Omega(n^k)$  different bottleneck matchings with the same longest edge, and a matching may be bottleneck along more than one quadratic piece of  $\mathcal{E}$ . In addition, the regions of the minimization diagram of  $\mathcal{E}$  might be non-convex and even disconnected.

A standard approach to break ties between bottleneck matchings for a given position in order to be more sensitive to the geometry of the point sets is to consider a lexicographic version. That is, among the matchings whose longest edge is as short as possible, consider those whose second longest edge is as short as possible, and so on. In order to give a formal definition, we recall that the *lexicographic order* on  $\mathbb{R}^k$  is defined as the total order induced by the relation  $(x_1, \dots, x_k) \preceq (y_1, \dots, y_k)$  if and only if there exists an  $m \in [k]$  such that  $x_i = y_i$  for all  $i < m$ , and  $x_m < y_m$ , or  $(x_1, \dots, x_k) = (y_1, \dots, y_k)$ .

**Definition 2.** Let  $A, B \subset \mathbb{R}^d$  be two finite point sets with  $k = |B| \leq |A|$ . The lex-bottleneck cost of a matching  $\sigma$  is the function  $g_\sigma : \mathbb{R}^d \rightarrow \mathbb{R}^k$  where the  $i$ -th coordinate of  $g_\sigma(t)$  is given by the length of the  $i$ -th longest edge of  $\sigma$  for  $B + t$  and  $A$ . A lex-bottleneck matching for  $A$  and  $B$  is a matching  $\pi$  such that  $g_\pi(0) \preceq g_\sigma(0)$  for every other matching  $\sigma$ .



**Fig. 1.** A label-induced coarsening of a bottleneck diagram for  $A$  and  $B$

Observe that, despite the assumed general position of  $A$  and  $B$ , the lex-bottleneck matching is not unique everywhere. However, we will prove later that it is unique for all translations except for a nowhere-dense set. Due to the non-uniqueness issues, the definition of the Voronoi-type diagram associated to bottleneck and lex-bottleneck matchings needs some care. We define them as polyhedral complexes, which simplifies traversing and optimizing in faces of the diagram. Hereafter, only full-dimensional faces will be called *cells*.

**Definition 3.** A bottleneck partial-matching Voronoi diagram (for short, bottleneck diagram) for point sets  $A, B \subset \mathbb{R}^d$  is a polyhedral complex  $\mathcal{T}$  covering  $\mathbb{R}^d$  and such that for every cell  $C$  of  $\mathcal{T}$  there is at least one matching  $\pi_C$  such that  $f_{\pi_C}(t) \leq f_{\sigma}(t)$  for all  $t \in C$  and all matchings  $\sigma$ . A bottleneck labeling of this diagram is a function mapping each cell of  $\mathcal{T}$  to one such matching.

Figure 1 shows a pair of point sets and a coarsening of a bottleneck diagram for them (neighboring cells with the same label have been merged). Observe that for  $B = \{b\}$  the Voronoi diagram of  $A + b$  is a bottleneck diagram.

**Definition 4.** A lex-bottleneck partial-matching Voronoi diagram (for short, lex-bottleneck diagram) for point sets  $A, B \subset \mathbb{R}^d$  is a polyhedral complex  $\mathcal{T}$  covering  $\mathbb{R}^d$  and such that for every face  $c$  of  $\mathcal{T}$  there is at least one matching  $\pi_c$  such that  $g_{\pi_c}(t) \preceq g_{\sigma}(t)$  for all  $t$  in the relative interior of  $c$  and all matchings  $\sigma$ . A lex-bottleneck labeling of this diagram is a function mapping each face of  $\mathcal{T}$  to one such matching.

Note that the lex-bottleneck label of a cell is not necessarily a lex-bottleneck matching on its boundary, which justifies the previous definition.

Since any lex-bottleneck diagram is a bottleneck diagram, we prove some properties for the first, more restrictive type.

**Proposition 1.** For any pair of point sets  $A, B \subset \mathbb{R}^d$  with  $k = |B| \leq |A| = n$ , there exists a lex-bottleneck diagram of complexity  $O(n^{2d} k^{2d})$ .

*Proof.* The edge  $ab \in A \times B$  for a position  $t \in \mathbb{R}^d$  of  $B$  has (squared) length

$$\|b + t - a\|^2 = \|t\|^2 + \|b - a\|^2 + 2 \langle t, b - a \rangle.$$

For a pair of edges  $ab, a'b' \in A \times B$ , let  $h(ab, a'b')$  be the locus of points  $t \in \mathbb{R}^d$  for which  $\|b + t - a\|^2 = \|b' + t - a'\|^2$  or, equivalently, for which

$$2 \langle t, b - a - (b' - a') \rangle = \|b' - a'\|^2 - \|b - a\|^2.$$

By our general position assumption on  $A$  and  $B$ , this defines a hyperplane in  $\mathbb{R}^d$ . Let  $\mathcal{H}$  be the arrangement induced by all these hyperplanes  $h(ab, a'b')$  and let  $c$  be a face of  $\mathcal{H}$ . The order (and the ties) of the values  $\|b + t - a\|$ ,  $ab \in A \times B$ , does not change when  $t$  is restricted to be a point in the relative interior of  $c$ . Hence, the set of lex-bottleneck matchings is the same for all such points and any matching in the set can thus be used as a label for  $c$ . Since  $\mathcal{H}$  is induced by  $O(n^2k^2)$  hyperplanes in  $\mathbb{R}^d$ , its complexity is  $O((n^2k^2)^d)$  (cf. [11]).  $\square$

Motivated by the construction in the previous proof, for given finite point sets  $A, B \subset \mathbb{R}^d$ , we define  $\mathcal{H}(A, B)$  to be the arrangement of the hyperplanes

$$h(ab, a'b') = \{t \in \mathbb{R}^d : \|b + t - a\| = \|b' + t - a'\|\},$$

for  $ab, a'b' \in A \times B$ . In order to show that there are some hyperplanes of  $\mathcal{H}(A, B)$  that we can safely ignore, we introduce some technical definitions and notation.

**Definition 5.** Let  $A, B \subset \mathbb{R}^d$  be two finite point sets with  $k = |B| \leq |A|$ . Given  $t \in \mathbb{R}^d$  and  $b \in B$ , we define the neighborhood  $E_b(t) \subseteq A \times \{b\}$  of  $b$  at  $t$  as the set including an edge  $ab$  if and only if  $\|b + t - a\|$  is among the  $k$  smallest values in  $\{\|b + t - a'\| : a' \in A\}$ . Let  $E(t) = \cup_{b \in B} E_b(t)$ . A set  $E \subseteq E_b(t)$  is a minimal set for  $b$  at  $t$  if  $|E| = k$  and  $\|b + t - a\| \leq \|b + t - a'\|$  for all  $ab \in E$  and all  $a'b \in A \times \{b\} \setminus E$ . Let  $X \subset \mathbb{R}^d$  be the (dense) set consisting of points  $t \in \mathbb{R}^d$  such that  $|E_b(t)| = k$  for all  $b \in B$ . An edge  $ab \in A \times B$  is called relevant if there exists  $t \in X$  such that  $ab \in E_b(t)$ . Let  $\mathcal{R}(A, B)$  be the arrangement induced by  $h(ab, a'b')$ , with both  $ab, a'b' \in A \times B$  relevant.

**Proposition 2.** Let  $A, B \subset \mathbb{R}^d$  be finite point sets. The arrangement  $\mathcal{R}(A, B)$  is a lex-bottleneck diagram for  $A$  and  $B$ . Furthermore, there is a unique lex-bottleneck matching for every point interior to a cell of this arrangement.

*Proof.* Note first that if  $E(t)$  contains a minimal set at  $t$  for all  $b \in B$ , then  $E(t)$  contains a lex-bottleneck matching for  $B + t$  and  $A$ . Moreover, if  $|E_b(t)| = k$  for all  $b \in B$ , then every bottleneck matching is contained in  $E(t)$ .<sup>1</sup>

Observe that  $E_b(t_1) = E_b(t_2)$  for every pair of points  $t_1, t_2$  interior to a cell  $C$  of  $\mathcal{R}(A, B)$ , and that  $|E_b(t_1)| = k$ , for all  $b \in B$ . Thus, any lex-bottleneck matching for a point interior to  $C$  is contained in  $E(t_1)$ . Moreover, since the lengths of the edges in  $E(t)$  are distinct and their ordering is the same for all  $t$  interior to  $C$ , there is a unique way to label  $C$ .

<sup>1</sup> Both claims will be implied by observations in the next section.

Let  $t_0$  be a point in a lower-dimensional face  $c$  of  $\mathcal{R}(A, B)$ , and let  $C \supset c$  be a cell. For continuity reasons, if  $t$  is interior to  $C$ , then  $E_b(t)$  is a minimal set for  $b$  at  $t_0$ , which implies that there is a lex-bottleneck matching for  $t_0$  in  $E(t)$ . Consequently, for every translation there is a lex-bottleneck matching using only relevant edges. Since in the relative interior of every face of  $\mathcal{R}(A, B)$ , the order (considering also the ties) of the lengths of the relevant edges is fixed, every face can be labeled with a lex-bottleneck matching.  $\square$

The following result used in [3] will allow us to improve the bound of Proposition 1 in low dimensions.

**Lemma 1 ([3]).** *Let  $A \subset \mathbb{R}^2$  be a set of  $n$  points. There are  $O(nk)$  bisectors that support all edges of  $j$ -th order Voronoi diagrams of  $A$  for all  $j \leq k$ .*

**Theorem 1.** *Given  $A, B \subset \mathbb{R}^2$  (resp.  $A, B \subset \mathbb{R}$ ) with  $k = |B| \leq |A| = n$ , there is a lex-bottleneck diagram for  $A$  and  $B$  of complexity  $O(n^2k^6)$  (resp.  $O(nk^3)$ ).*

*Proof.* Given a cell  $C$  of  $\mathcal{R}(A, B)$ , let  $E(C)$  denote  $E(t)$  for any point  $t$  interior to  $C$ . We say that an edge  $e$  of  $\mathcal{R}(A, B)$  between two cells  $C_l$  and  $C_r$  uses a bisector  $h = h(ab, a'b')$  if  $h$  supports  $e$  and  $ab, a'b' \in E(C_l) \cup E(C_r)$ . As seen in the proof of Proposition 2, there is a lex-bottleneck labeling of  $\mathcal{R}(A, B)$  that uses in every face  $c$  only edges from  $E(C)$  with  $C$  a cell containing  $c$ . Therefore, if no edge uses a bisector  $h$ , then  $h$  can be omitted from  $\mathcal{R}(A, B)$ ; the resulting arrangement will still be a lex-bottleneck diagram.

Let  $h(ab, a'b)$  be a bisector used by an edge  $e = C_l \cap C_r$ , and consider the point set  $S(b) = A - b$ . Using that the locus of points in the plane at the same distance from three different given points is either a point or the empty set, it follows that for any point  $t$  interior to  $e$  only  $a - b$  and  $a' - b$  are at distance  $\|b + t - a\|$  from  $t$ , and the set of points of  $S(b)$  closer than  $a - b$  is the same. In addition, since  $ab$  and  $a'b$  are relevant, the aforementioned set has cardinality at most  $k - 1$ . Thus,  $e$  must be supported by a  $j$ -th order Voronoi edge of  $S(b)$  for some  $j \leq k$ .

Let  $h(ab, a'b')$  with  $b \neq b'$  be a bisector used by an edge  $e = C_l \cap C_r$ , and consider the point set  $S(b, b') = (A - b) \sqcup (A - b')$ . Note that the previous union is disjoint because of our general-position assumption. Simple algebraic manipulations show that  $t \in \mathbb{R}^2$  is closer to  $a_1 - b$  than to  $a_2 - b'$  if and only if  $b + t$  is closer to  $a_1$  than  $b' + t$  is to  $a_2$ , for any choice of  $a_1, a_2 \in A$ . The observation in the previous paragraph implies that for a point  $t$  in the relative interior of  $e$ , the only points in  $S(b, b')$  at distance  $\|b + t - a\|$  from  $t$  are  $a - b$  and  $a' - b'$ . In addition, since both edges are relevant, the (open) disk centered at  $t_0$  and through  $a - b$  and  $a' - b'$  contains at most  $k - 1$  points of  $A - b$  and at most  $k - 1$  points of  $A - b'$ . Thus, the bisector  $h(ab, a'b')$  supports a  $j$ -th order Voronoi edge of  $S(b, b')$  for some  $j \leq 2k - 1$ .

Applying Lemma 1 to  $S(b)$  for all  $b \in B$  and to  $S(b, b')$  for every pair  $b, b' \in B$ , it follows that the number of bisectors that are used by some edge is  $O(k^2 \cdot nk)$ . The complexity of the arrangement of these bisectors is thus  $O(n^2k^6)$ . We refer henceforth to this arrangement as  $\mathcal{L}(A, B)$ .

The case of the line is proven analogously.  $\square$

Based on a one-dimensional example of Rote [17], one may derive a lower bound on the complexity of any lex-bottleneck diagram.

**Proposition 3.** *Given  $k, n \in \mathbb{N}$  with  $n \geq k \geq d$ , there exist point sets  $A, B \subset \mathbb{R}^d$  with  $|B| = k$  and  $|A| = n$  such that any lex-bottleneck diagram for  $A$  and  $B$  has complexity  $\Omega(k^d(n - k)^d)$ .*

### 3 Construction of Bottleneck Diagrams

We first discuss algorithmic techniques to construct an unlabeled bottleneck or lex-bottleneck diagram for a pair of point sets in the plane.

**Lemma 2.** *A lex-bottleneck diagram for any given pair of point sets  $A, B \subset \mathbb{R}^2$  with  $k = |B| \leq |A| = n$  can be constructed in  $O(n^2k^6)$  time.*

*Proof.* We construct the arrangement  $\mathcal{L}(A, B)$  of bisectors supporting some edge of the  $j$ -th order Voronoi diagram of  $S(b)$  for all  $b \in B$  and all  $j \leq k$ , and the bisectors supporting some edge of the  $j$ -th order Voronoi diagram of  $S(b, b')$  for all pairs  $b, b' \in B$  and all  $j \leq 2k - 1$ , as defined in the proof of Theorem 1. To select the bisectors generated by each of these point sets, we use an algorithm by Chan [7] that constructs the facial structure of the ( $\leq s$ )-level of an arrangement of  $m$  planes in  $\mathbb{R}^3$  in  $O(m \log m + ms^2)$  expected time or  $O(ms^2(\log m / \log s)^{O(1)})$  deterministic time. This requires  $O(m)$  space using the data structure in [1]. We can thereby construct the  $O(k^2)$  necessary structures and discover the bisectors belonging to  $\mathcal{L}(A, B)$ . We then compute the arrangement of these  $O(nk^3)$  lines in  $O((nk^3)^2)$  time using, for instance, the incremental algorithm in [8].  $\square$

Our aim is now to find a bottleneck labeling of  $\mathcal{L}(A, B)$ . The problem of finding a bottleneck matching for a fixed position of  $B$  can be translated into a matching problem in a weighted bipartite graph on  $A$  and  $B$ , where the weight of an edge from  $a \in A$  to  $b \in B$  is the Euclidean distance between the corresponding points. Therefore, we first introduce the necessary notation on bottleneck assignments in bipartite graphs and discuss the known methods that are relevant for our purposes.

Let  $G = (U, V; E)$  be a bipartite graph with edge set  $E$  and vertex set partitioned into components  $U$  and  $V$  of sizes  $k = |U| \leq |V| = n$ . In this section, the notion of *matching* in  $G$  will be relaxed to admit any set of pairwise-disjoint edges  $\sigma \subseteq E$ . Given a matching  $\sigma$ , a vertex belonging to some edge of  $\sigma$  is called a *matched vertex*. A matching is *complete* if all the vertices in  $U$  are matched. An *alternating path* for  $\sigma$  is a path in  $G$  (with no repeated vertices) such that the even edges are in  $\sigma$  and the odd ones are in  $E \setminus \sigma$ . An *augmenting path* for  $\sigma$  is an alternating path starting and ending at non-matched vertices. Note that, if  $\gamma$  is an augmenting path for  $\sigma$ , the matching  $\tau = (\sigma \setminus \gamma) \cup (\gamma \setminus \sigma)$  has one more matched vertex than  $\sigma$ .

As in the geometric setting, we simplify notation by denoting the edge  $\{u, v\} \in E$  by  $uv$ , and we identify a complete matching in  $G$  with the injection of  $U$  into  $V$  it induces. We assume that  $G$  is complete and that its edges

have weights given by a function  $w : E \rightarrow \mathbb{R}_{\geq 0}$ . For our purposes, there is no loss of generality in assuming that the weights are integers, thus spending  $O(nk \log n)$  extra time, we could sort them and replace the weight of an edge by its position in the sorted list. An injection  $\pi : U \hookrightarrow V$  is called a *bottleneck matching* in  $G$  with respect to  $w$  if

$$\max_{i \in U} w(i\pi(i)) \leq \max_{i \in U} w(i\sigma(i)),$$

for every injection  $\sigma : U \hookrightarrow V$ .

The lexicographic bottleneck assignment problem was introduced by Burkard and Rendl in [6]. Analogously to the geometric setting, the cost of a complete matching  $\pi : U \hookrightarrow V$  in this problem depends on the values  $w(u\pi(u))$  for  $u \in U$ : a *lexicographic-bottleneck matching* in  $G$  with respect to  $w$  is an injection minimizing the cost when the edge weights are compared lexicographically.

Given  $r \in [kn]$ , we define  $G(r)$  to be the graph that remains after removing the edges from  $G$  whose weight is not among the  $r$  smallest. An important observation is that a complete matching  $\sigma$  in  $G(r)$  is bottleneck if and only if  $G(r-1)$  has no complete matching.

We now assume that  $G$  is bipartite complete, and hence we know that it contains a complete matching. Analogously to the geometric case, given  $u \in U$ , we say that a set  $E_u \subseteq \{u\} \times V$  of  $k$  edges is a *minimal set for  $u$*  if no edge in  $\{u\} \times V \setminus E_u$  has weight smaller than an edge in  $E_u$ . Therefore, if a set  $E \subseteq V \times U$  contains a minimal set  $E_u$  for every  $u \in U$ , it must contain a lexicographic-bottleneck matching. Indeed, if a matching uses an edge  $uv$  that is not in  $E_u$ , this edge can be replaced by an edge of  $E_u$ , since at least one of these edges is unmatched. The resulting matching has better or equal lexicographic cost. We may restrict then the attention to a set of  $k^2$  edges, after sorting the edges incident to each vertex in  $O(nk \log n)$  time. For a graph with  $k^2$  edges and whose matching of maximum cardinality is of size  $k$ , the Gabow-Tarjan algorithm [13] finds a bottleneck matching in  $O(k^2 \sqrt{k} \log k)$  time, according to the analysis in [5]. A lexicographic-bottleneck assignment can be computed in  $O(k^4)$  time by combining the approach in [18] with the algorithm for the linear sum assignment problem proposed in [16].

In order to find a bottleneck labeling of  $\mathcal{L}(A, B)$  we will traverse it maintaining relevant information to update the matching. To this end, we need some insight into the dynamic behavior of bottleneck matchings. The following lemma, whose proof must be postponed to a full version of this work, analyzes how a bottleneck matching is affected by local changes.

**Lemma 3.** *Let  $G = (U, V; E, w)$  be a bipartite graph with  $w : E \rightarrow [|E|]$  giving weights to its edges. Let  $\mu$  be a bottleneck matching for  $G$ , and let  $l \in E$  be the longest edge of  $\mu$  in  $G$ . For a fixed  $j \in [|E| - 1]$ , let  $G' = (U, V; E, w')$  where  $w'$  coincides with  $w$  except that  $w'(e) = j$  if  $w(e) = j + 1$  and  $w'(e') = j + 1$  if  $w(e') = j$ , for all  $e \in E$ .*

1. *If  $w(l) \notin \{j, j + 1\}$ , then  $\mu$  is a bottleneck matching for  $G'$ .*
2. *If  $w(l) \in \{j, j + 1\}$  and  $G'(j)$  does not have a complete matching, then  $\mu$  is a bottleneck matching for  $G'$ .*

3. If  $w(l) \in \{j, j+1\}$  and  $G'(j)$  has a complete matching  $\nu$ , then  $\nu$  is a bottleneck matching for  $G'$ .

We are now ready to prove the main result of this section.

**Theorem 2.** *Given  $A, B \subset \mathbb{R}^2$  with  $k = |B| \leq |A| = n$ , a bottleneck diagram for  $A$  and  $B$  (and a labeling of it) can be computed in  $O(n^2k^8)$  time and a (labeled) lex-bottleneck diagram in  $O(n^2k^{10})$  time.*

*Proof.* We first construct the arrangement  $\mathcal{L}(A, B)$  as in Lemma 2, making sure that we remember for every selected line which are the two edges of  $A \times B$  with the same length on the line. We pick a point  $t$  interior to an arbitrary cell  $C$  of  $\mathcal{L}(A, B)$  and sort, for every  $b \in B$ , the values  $\{\|b + t - a\| : a \in A\}$ . Then, we construct the set  $E(t)$  of  $k^2$  edges, and the weight function  $w$  representing the order of the edges in  $E(t)$  according to their lengths.

For the construction of a bottleneck labeling, we first find a bottleneck matching  $\mu$  in  $G = (A, B; E(t), w)$  in  $O(k^2\sqrt{k \log k})$  time using the Gabow-Tarjan algorithm. Since  $E(t)$  contains a minimal set for every point in  $B$ , the matching will be bottleneck for the complete graph as well. We then traverse  $\mathcal{L}(A, B)$  maintaining the graph  $G$  for a point  $t$  interior to the current cell and  $\mu$  as a bottleneck matching for  $G$ . More precisely, when a bisector of  $\mathcal{L}(A, B)$  involving edges  $e_1 = ab$  and  $e_2 = a'b$  with  $e_2 \in E(t)$ ,  $e_1 \notin E(t)$  is traversed, we replace  $e_2$  with  $e_1$  (weighted by  $w(e_2)$ ) in  $G$ . In the other cases, we only need to swap the weights of the corresponding edges if both belong to  $E(t)$ . We update the bottleneck matching as indicated in Lemma 3. In order to do so, we might need to check whether the “new”  $G(j)$  has a complete matching for the corresponding  $j \in [k^2]$ . Note that, if  $e$  is the edge increasing its length and  $\mu$  is an “old” bottleneck matching,  $\mu \setminus e$  is a matching in the new  $G(j)$  of size at least  $k - 1$ . If its size is  $k - 1$ , Berge’s lemma [4] ensures that there is a complete matching in  $G(j)$  if and only if there is an augmenting path for  $\mu \setminus e$  in  $G(j)$ , which can be checked in  $O(k^2)$  time. Note that, even under our general-position assumption, the bisectors of several different pairs of edges might coincide supporting a single edge of  $\mathcal{L}(A, B)$ . Fortunately, such multiple edges can be handled by sequentially performing simple updates as described above. In addition, if one looks into the proof of Lemma 1 based on the Clarkson-Shor technique [9], it is clear that the bound holds even if the bisectors are counted with multiplicities (i.e., if the bisectors defined by  $m$  pairs of points happen to coincide, the line is counted  $m$  times). This works because the bound used for the number of bisectors contributing to a Voronoi diagram is actually a bound for its number of edges. Therefore, the bound of  $O(n^2k^6)$  on the complexity of  $\mathcal{L}(A, B)$  holds even if the coincident bisectors are infinitesimally perturbed to be parallel, distinct lines. Since the number of simple updates is bounded by the number of edges in this perturbed arrangement, the traversal requires  $O(k^2 \cdot n^2k^6)$  time.

For the lex-bottleneck labeling, we attain the claimed bound by maintaining the graph  $G$  as before and computing a label from scratch in every face (where  $w$  might need to be modified in order to indicate ties) of  $\mathcal{L}(A, B)$  in  $O(k^4)$  time as noted before. □



In higher dimensions, the bounds on the complexity of the arrangement  $\mathcal{L}(A, B)$  and on the running time for its construction resulting from extending our results in the plane are not better than the simpler approach of constructing and labeling  $\mathcal{H}(A, B)$ . Our algorithms can be adapted without difficulties to this end, leading to the following result.

**Theorem 3.** *Let  $A, B \subset \mathbb{R}^d$  be two point sets with  $k = |B| \leq |A| = n$ . There is a lex-bottleneck diagram of complexity  $O(n^{2d}k^{2d})$ . A bottleneck labeling for this diagram can be computed in  $O(n^{2d}k^{2d+2})$  time, and a lex-bottleneck labeling in  $O(n^{2d}k^{2d+4})$  time.*

### 4 Applications

We conclude our investigations by discussing applications of bottleneck diagrams. For convenience of presentation, we restrict ourselves to problems in the plane.

A bottleneck diagram for point sets  $A$  and  $B$  is a useful tool to solve the bottleneck partial-matching problem and provide, in addition, the translation minimizing the distance. More precisely, we will find a translation  $t^* \in \mathbb{R}^2$  and an injection  $\pi : B \hookrightarrow A$  such that  $\mathcal{E}(t^*) = \min_{t \in \mathbb{R}^2} \mathcal{E}(t) = f_\pi(t^*)$ .

**Corollary 1.** *The bottleneck partial-matching problem for point sets  $A, B \subset \mathbb{R}^2$  with  $k = |B| \leq |A| = n$  can be solved in  $O(n^2k^8)$  time.*

*Proof.* We first construct  $\mathcal{L}(A, B)$  and a labeling for it in time  $O(n^2k^8)$  as described in the proof of Theorem 2. We then traverse the arrangement optimizing the function  $f(t) = \|b + t - a\|^2$  in every (convex) cell keeping the minimum throughout the diagram, where  $ab$  is the longest edge of the bottleneck matching that labels the current cell. Let  $C$  be a cell in  $\mathcal{L}(A, B)$  and let  $t_0 = a - b$  be the global minimum of the function  $f(t)$ . If  $t_0 \in C$ , obviously  $f(t_0) = 0$  is the global minimum we are after. Otherwise, the minimum is attained at the intersection of an edge of the cell with the line perpendicular to it through  $t_0$  (if it is non-empty) or at a vertex of  $C$ . Note that during the traversal every edge will be examined twice and every vertex once for every cell containing it. Thus, the total time needed to perform the mentioned optimization in every cell is proportional to the complexity of the diagram.  $\square$

We consider now the problem of finding a motion for  $B$  from an initial position to a final position such that the maximum bottleneck value (as defined in Definition 1) attained during the motion is minimized.

**Definition 6.** *The bottleneck value of a curve  $\gamma : [0, 1] \rightarrow \mathbb{R}^2$  with respect to point sets  $A, B \subset \mathbb{R}^2$  with  $k = |B| \leq |A| = n$  is*

$$F(\gamma) = \max_{s \in [0, 1]} \mathcal{E}(\gamma(s)) = \max_{s \in [0, 1]} \min_{\sigma : B \hookrightarrow A} \max_{b \in B} \|b + \gamma(s) - \sigma(b)\|^2.$$

*The curve  $\gamma$  is called a bottleneck path if  $F(\gamma) \leq F(\varphi)$  for every other curve  $\varphi : [0, 1] \rightarrow \mathbb{R}^2$  with  $\varphi(0) = \gamma(0)$  and  $\varphi(1) = \gamma(1)$ .*

A bottleneck path between two positions can be useful in motion planning where the points of  $A$  represent fixed anchor points and the points of  $B$  represent the position of articulations of a moving robot. The dual graph of the arrangement  $\mathcal{L}(A, B)$  contains the necessary information to compute a bottleneck path from any initial position to any final position.

**Definition 7.** *The bottleneck graph of two finite point sets  $A, B \subset \mathbb{R}^2$  is the weighted graph  $\mathcal{L}(A, B)^*$  dual to  $\mathcal{L}(A, B)$ , where an edge  $e^*$  of the graph dual to an edge  $e$  of  $\mathcal{L}(A, B)$  has weight  $\min_{t \in e} \mathcal{E}(t)$ .*

Via the bottleneck graph we can now characterize the existence of a path of given bottleneck value.

**Lemma 4.** *Let  $t_0, t_1 \in \mathbb{R}^2$  and  $\delta \in \mathbb{R}$ . Let  $C_0$  and  $C_1$  be cells of  $\mathcal{L}(A, B)$  such that  $t_0 \in C_0$  and  $t_1 \in C_1$ . There is a path with bottleneck value at most  $\delta$  from  $t_0$  to  $t_1$  if and only if  $\mathcal{E}(t_0), \mathcal{E}(t_1) \leq \delta$  and there is a path from  $C_0^*$  to  $C_1^*$  in  $\mathcal{L}(A, B)^*$  whose longest edge has weight at most  $\delta$ .*

*Proof.* Observe that in every cell of  $\mathcal{L}(A, B)$  there is a bottleneck matching whose cost coincides with  $\mathcal{E}$  in the cell. By definition,  $\mathcal{E}$  is a convex function in every such (convex) cell. Hence, assuming that  $C_0 = C_1$ , the line segment joining  $t_0$  and  $t_1$  has bottleneck value  $\max\{\mathcal{E}(t_0), \mathcal{E}(t_1)\}$  and no path can attain a smaller value. We assume now that  $C_0 \neq C_1$  and let  $\gamma$  be any path from  $t_0$  to  $t_1$ . We can replace each of the connected arcs of  $\gamma$  entering a cell  $C$  of  $\mathcal{L}(A, B)$  in a point  $t_{\text{in}}$  and leaving it in a point  $t_{\text{out}}$  by the line segment joining these two points without increasing the bottleneck value of the path. Again, we do not increase the bottleneck value of the path when we substitute this line segment by the one joining the points  $t_{\text{in}}^*$  and  $t_{\text{out}}^*$ , where  $t_{\text{in}}^*$  is the point with minimum bottleneck value on the edge of  $C$  that contains  $t_{\text{in}}$ , and  $t_{\text{out}}^*$  is the one attaining the minimum value on the edge containing  $t_{\text{out}}$ . Similarly, the parts of the path in  $C_0$  and  $C_1$ , starting at  $t_0$  and ending at  $t_1$ , respectively, can be replaced with the line segment from  $t_0$  (or  $t_1$ ) to the point attaining the minimum of  $\mathcal{E}$  on whichever edge of  $C_0$  (or  $C_1$ ) the path crosses first (or last).

The previous observations imply that a bottleneck path is among the polygonal paths whose vertices (except for  $t_0$  and  $t_1$ ) lie on the minima of  $\mathcal{E}$  along edges of  $\mathcal{L}(A, B)$ . The bottleneck value of such a path is the maximum of the weights of the edges in the corresponding path in  $\mathcal{L}(A, B)^*$  and the values  $\mathcal{E}(t_0)$  and  $\mathcal{E}(t_1)$ . □

**Theorem 4.** *Given  $t_0, t_1 \in \mathbb{R}^2$ , a bottleneck path from  $t_0$  to  $t_1$  with respect to  $A, B \subset \mathbb{R}^2$  with  $k = |B| \leq |A| = n$  can be computed in time  $O(n^2 k^6 (k^2 + \log n))$ .*

*Proof.* We first compute the arrangement  $\mathcal{L}(A, B)$  and the associated bottleneck graph  $\mathcal{L}(A, B)^*$  in time  $O(n^2 k^8)$  by Theorem 2. The number of edges and vertices of  $\mathcal{L}(A, B)^*$  is  $O(n^2 k^6)$  due to Theorem 1 and the weights of its edges are all nonnegative. Therefore, the path with minimum bottleneck value in the graph can be found in  $O(n^2 k^6 \log n)$  time via the implementation of Dijkstra’s algorithm using heaps. By Lemma 4, the associated polygonal path is guaranteed to be a bottleneck path from  $t_0$  to  $t_1$ . □

## References

1. Afshani, P., Chan, T.M.: Optimal Halfspace Range Reporting in Three Dimensions. In: Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms, pp. 180–186 (2009)
2. Alt, H., Mehlhorn, K., Wagener, H., Welzl, E.: Congruence, similarity, and symmetries of geometric objects. *Discrete Comput. Geom.* **3**(1), 237–256 (1988)
3. Ben-Avraham, R., Henze, M., Jaume, R., Keszegh, B., Raz, O.E., Sharir, M., Tubis, I.: Minimum Partial-Matching and Hausdorff RMS-Distance under Translation: Combinatorics and Algorithms. In: Schulz, A.S., Wagner, D. (eds.) *ESA 2014*. LNCS, vol. 8737, pp. 100–111. Springer, Heidelberg (2014)
4. Berge, C.: Two Theorems in Graph Theory. *Proc. Nat. Acad. Sci. U.S.A.* **43**, 842–844 (1957)
5. Burkard, R.E., Dell’Amico, M., Martello, S.: *Assignment Problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (2009)
6. Burkard, R.E., Rendl, F.: Lexicographic bottleneck problems. *Oper. Res. Lett.* **10**(5), 303–308 (1991)
7. Chan, T.M.: Random Sampling, Halfspace Range Reporting, and Construction of ( $\leq k$ )-Levels in Three Dimensions. *SIAM J. Comput.* **30**(2), 561–575 (2000)
8. Chazelle, B., Guibas, L.J., Lee, D.T.: The power of geometric duality. *BIT* **25**(1), 76–90 (1985)
9. Clarkson, K.L., Shor, P.W.: Applications of Random Sampling in Computational Geometry. II. *Discrete Comput. Geom.* **4**(5), 387–421 (1989)
10. Croce, F.D., Paschos, V.T., Tsoukias, A.: An improved general procedure for lexicographic bottleneck problems. *Oper. Res. Lett.* **24**(4), 187–194 (1999)
11. Edelsbrunner, E. (ed.): *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science, vol. 10. Springer, Berlin (1987)
12. Efrat, A., Itai, A., Katz, M.J.: Geometry Helps in Bottleneck Matching and Related Problems. *Algorithmica* **31**(1), 1–28 (2001)
13. Gabow, H.N., Tarjan, R.E.: Algorithms for two bottleneck optimization problems. *J. Algorithms* **9**(3), 411–417 (1988)
14. Henze, M., Jaume, R.: Bottleneck Partial-Matching Voronoi Diagrams. In: *Proc. 30th European Workshop on Computational Geometry* (2014)
15. Henze, M., Jaume, R., Keszegh, B.: On the complexity of the partial least-squares matching Voronoi diagram. In: *Proc. 29th European Workshop on Computational Geometry*, pp. 193–196 (2013)
16. Ramshaw, L., Tarjan, R.E.: On minimum-cost assignments in unbalanced bipartite graphs. Technical report, HP Labs Technical Report HPL-2012-40 (June 2012)
17. Rote, G.: Partial Least-Squares Point Matching under Translations. In: *Proc. 26th European Workshop on Computational Geometry*, pp. 249–251 (2010)
18. Sokkalingam, P.T., Aneja, Y.P.: Lexicographic bottleneck combinatorial problems. *Oper. Res. Lett.* **23**(1–2), 27–33 (1998)

# Ham-Sandwich Cuts for Abstract Order Types

Stefan Felsner<sup>1</sup> and Alexander Pilz<sup>2</sup>(✉)

<sup>1</sup> Institut für Mathematik, Technische Universität Berlin, Berlin, Germany

`felsner@math.tu-berlin.de`

<sup>2</sup> Institute for Software Technology, Graz University of Technology, Graz, Austria

`apilz@ist.tugraz.at`

**Abstract.** The linear-time ham-sandwich cut algorithm of Lo, Matoušek, and Steiger for bi-chromatic finite point sets in the plane works by appropriately selecting crossings of the lines in the dual line arrangement with a set of well-chosen vertical lines. We consider the setting where we are not given the coordinates of the point set, but only the orientation of each point triple (the order type) and give a deterministic linear-time algorithm for the mentioned sub-algorithm. This yields a linear-time ham-sandwich cut algorithm even in our restricted setting. We also show that our methods are applicable to abstract order types.

## 1 Introduction

Goodman and Pollack investigated ways of partitioning the infinite number of point sets in the plane into a finite number of equivalence classes. To this end they introduced *circular sequences* [13] and *order types* [14]. Two point sets  $S_1$  and  $S_2$  have the same *order type* iff there exists a bijection between the sets s.t. a triple in  $S_1$  has the same orientation (clockwise or counterclockwise) as the corresponding triple in  $S_2$  (in this paper we only consider point sets in general position, i.e., no three points are collinear).

The order type determines many important properties of a point set, e.g., its convex hull and which spanned segments cross. Determining the orientation of a point triple (called a *sidedness query*) can be done in a computationally robust way [4]. Therefore, algorithms that base their decisions solely on sidedness queries allow robust implementations [6]. Furthermore, this restriction is helpful for mechanically proving correctness of algorithms [23, 24].

The duality between point sets and their dual line arrangements is a well-established tool in discrete and computational geometry. Line arrangements can be generalized to pseudo-line arrangements, and many combinatorial and algorithmic questions that can be asked for line arrangements are also interesting for pseudo-line arrangements. The order type of a point set is encoded in the structure of the dual line arrangement of a point set, in particular by the lines

---

Supported by the ESF EUROCORES programme EuroGIGA - ComPoSe. A.P. is supported by Austrian Science Fund (FWF): I 648-N18. Part of this work was presented in the PhD thesis [25] of the second author.

(and even by the number of lines [14]) above and below a crossing in the dual arrangement. See [16] for an in-depth treatment of that topic. The orientation of a triple of pseudo-lines can be obtained from the ordering of crossings just as for lines. The triple-orientations fulfill certain axioms, and concepts like the convex hull can be defined for sets with appropriate triple-orientations [17] even though they may not be realizable by a point set. Such a generalization of order types is known as *abstract order type*. Besides their combinatorial properties, algorithmic aspects of abstract order types have been studied. Knuth devotes a monograph [17] to this generalization and its variants, in particular w.r.t. convex hulls. Motivated by Knuth's open problems, Aichholzer, Miltzow, and Pilz [3] show how to obtain, for a given pair  $(a, b)$  of an abstract order type, the edges of the convex hull that are intersected by the supporting line of  $ab$  in linear time, using only sidedness queries. There appears to be no known reasonable algorithmic problem that can be formulated for both order types and abstract order types such that there is an algorithm for order types that is asymptotically faster than any possible algorithm for abstract order types (see also the discussion in [11, p. 29]). In this paper, we show that the ham-sandwich cut is another problem that does not provide such an example. Apart from being of theoretical interest, abstract order types that are not realizable by point sets occur naturally when the point set is surrounded by a simple polygon and point triples are oriented w.r.t. the geodesics between them [2].

Given a pair  $(a, b)$  of points of a bi-chromatic point set  $S$  of  $n$  points that are either red or blue, the supporting line of  $a$  and  $b$  is a *ham-sandwich cut* if not more than half of the red and half of the blue points are on either side of  $ab$ . This can be verified by using only sidedness queries (implying a brute-force algorithm running in  $\Theta(n^3)$  time). Megiddo [22] presented a linear-time algorithm for the case in which the points of one color are separable from the points of the other color by a line. Edelsbrunner and Waupotitsch [10] gave an  $O(n \log(\min\{n_r, n_b\}))$  time algorithm for the general case, with  $n_r$  red and  $n_b$  blue points. Eventually, a linear-time algorithm was provided by Lo, Matoušek, and Steiger [18] for the general setting (abbrev. *LMS algorithm*). Their approach generalizes to arbitrary dimensions. Bose et al. [7] generalize ham-sandwich cuts to points inside a simple polygon, obtaining a randomized  $O((n+m) \log r)$  time algorithm, where  $m$  is the number of vertices of the polygon, of which  $r$  are reflex. Ham-sandwich cuts belong to a class of problems in computational geometry that deal with partitioning finite sets of points by hyperplanes while imposing constraints on both the subsets of the partition as well as on the hyperplanes; see, e.g., [26] for algorithms for related problems.

The LMS algorithm works on the dual line arrangement of the point set and has to solve the following sub-problem.

*Problem 1.* Given a line arrangement  $\mathcal{A}$  in the plane and two lines  $p$  and  $q$  of that arrangement, let  $v$  be the vertical line passing through the crossing of  $p$  and  $q$ . For a subset  $B$  of the lines in  $\mathcal{A}$  and an integer  $k \leq |B|$ , find a line  $m \in B$  such that the  $y$ -coordinate of the point  $v \cap m$  is of rank  $k$  in the sequence of  $y$ -coordinates of the finite point set  $v \cap \bigcup_{b \in B} b$ .

This problem can be solved in linear time by directly applying the linear-time selection algorithm [5] to the  $y$ -coordinates of the intersections of all lines in  $B$  with  $v$ . Clearly, the order of the intersections of lines with a vertical line at a crossing is not a property of the order type represented by the arrangement (e.g., in Fig. 1, one could change the order of the lines 2 and 4 above the crossing between line 5 and line 6). The order type only determines the set of lines above and below a crossing.

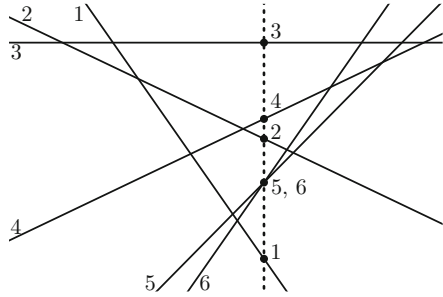


Fig. 1. Crossings along a vertical line

A reformulation of Problem 1 for abstract order types faces two problems. First, the vertical direction is not determined by the order type; there is, in general, an exponential number of different ways to draw a pseudo-line arrangement representing the abstract order type (w.r.t. the  $x$ -order of crossings), yielding too many different orders of the pseudo-lines along the vertical line through a crossing. Second, even when such a vertical line is given, directly applying the linear-time selection algorithm requires that a query comparing the order of two pseudo-lines on the vertical line can be answered in constant time.

In this paper, we show how to overcome these two problems. We define a “vertical” pseudo-line through each crossing in a pseudo-line arrangement and show how to select the pseudo-line of a given rank in the order defined by such a “vertical” pseudo-line. We give the precise definition in Section 2 where we also examine properties of the construction. The result is presented in terms of a (dual) pseudo-line arrangement in the Euclidean plane  $\mathbb{E}^2$ . However, in our model we are not given an explicit representation but are only allowed sidedness queries. In Section 3, we first explain how the queries about a pseudo-line arrangement can be mapped to sidedness queries, and then give a linear-time algorithm for selecting a pseudo-line with a given rank. Our result allows for replacing vertical lines in the LMS algorithm, showing that it also works for abstract order types. An analysis of the LMS algorithm under this aspect and all omitted proofs are given in the full version of this paper.

The observation that the LMS algorithm in principle also works for pseudo-line arrangements has been used by Bose et al. [7] for their randomized linear-time algorithm for geodesic ham-sandwich cuts. However, their pseudo-lines are given by (weakly)  $x$ -monotone polygonal paths with a constant number of edges. Hence, the intersection of such a path with a vertical line can be computed in constant time, like in the straight-line setting. Their randomized algorithm runs in  $O((n + m) \log r)$  time, where  $n$  is the number of red and blue points,  $m$  is the number of vertices of the polygon, of which  $r$  are reflex. Geodesic order types are a subset of abstract order types [2]. When applying a result from [1] to get, after  $O(m)$  preprocessing time, the orientation of each triple of points in a simple

polygon in  $O(\log r)$  time in combination with the ham-sandwich cut algorithm for abstract order types, we obtain a deterministic  $O(n \log r + m)$  time algorithm for geodesic ham-sandwich cuts “for free”. (Note that this does not contradict optimal worst-case behavior shown by Bose et al. [7] for their algorithm, as their analysis is parameterized by  $(n + m)$  and  $r$ .)

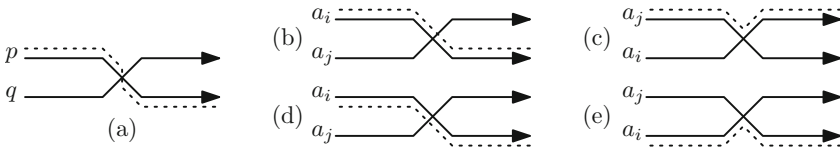
A *pseudo-line* is an  $x$ -monotone plane curve in  $\mathbb{E}^2$ . A *pseudo-line arrangement* is the cell complex defined by the dissection of  $\mathbb{E}^2$  by a set of pseudo-lines such that each pair of pseudo-lines intersects in exactly one point, at which they cross. An arrangement is *simple* if no three pseudo-lines intersect in the same point. Let  $\mathcal{A}$  be a simple arrangement of  $n$  pseudo-lines. The two vertically unbounded cells in  $\mathcal{A}$  are called the *north face* and the *south face*. The  $k$ -*level* of  $\mathcal{A}$  is the set of all points that lie on a pseudo-line of  $\mathcal{A}$  and have exactly  $k - 1$  pseudo-lines strictly above them. The level of a crossing  $pq$  is denoted by  $\text{lv}(pq)$ . The *upper envelope* of an arrangement is its 1-level, i.e., the union of the segments of pseudo-lines that are incident to the north face.

## 2 Pseudo-Verticals

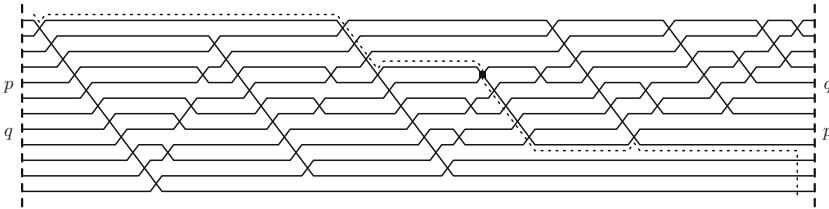
It will be convenient to consider all pseudo-lines being directed towards positive  $x$ -direction. Let  $p$  and  $q$  be two pseudo-lines in  $\mathcal{A}$  and let  $p$  start above  $q$ . We denote the latter by  $p \prec q$ . Our first aim is to define a pseudo-vertical through a crossing, i.e, an object that can be used like a vertical line through a crossing in our abstract setting.

For a crossing  $pq$  with  $p \prec q$  let  $\gamma_{pq}$  be a curve described by the following local properties. Initially,  $\gamma_{pq}$  passes through the crossing  $pq$  and enters the cell  $C$  directly above  $pq$ ; see Fig. 2 (a). To define  $\gamma_{pq}$  it is convenient to think of it as consisting of two parts. The *northbound ray* is the part starting at  $pq$  leading to the north face while the *southbound ray* connects  $pq$  to the south face. Starting from  $pq$  the northbound ray follows  $p$  against its direction moved slightly into the interior of cell  $C$ . In general the northbound ray of  $\gamma_{pq}$  will be slightly above some line  $a_i$  moving against the direction of  $a_i$ . When  $a_i$  is crossed by a pseudo-line  $a_j$  we have two cases. If  $a_i$  is crossed from below,  $\gamma_{pq}$  also crosses  $a_j$ , and continues following  $a_i$ ; see Fig. 2 (b). If  $a_i$  is crossed by  $a_j$  from above,  $\gamma_{pq}$  leaves  $a_i$  and continues following  $a_j$  against its direction; see Fig. 2 (c). This is continued until  $\gamma_{pq}$  follows some line  $a_i$  and there is no further crossing on  $a_i$  ahead (i.e., to the left). At that point, the northbound ray moves to the line at infinity and follows that line to the north face (i.e., it crosses all lines  $a$  with  $a \prec a_i$  in decreasing  $\prec$ -order). The southbound ray of  $\gamma_{pq}$  is defined in a similar manner. It follows some pseudo-lines in their direction but slightly below. It starts with  $p$  and when following  $a_i$  it changes to  $a_j$  at a crossing only when  $a_j$  is crossing from above (see Fig. 2 (d) and Fig. 2 (e)). The final part may again consist of some crossings with lines  $a$  with  $a \prec a_i$  in decreasing  $\prec$ -order.

We call  $\gamma_{pq}$  a *pseudo-vertical* and, in the following, identify several properties of such a curve. Note that, while we used the (rather informal) notion of “following” a pseudo-line,  $\gamma_{pq}$  is actually defined by the cells it traverses (i.e.,



**Fig. 2.** Local definition of a pseudo-vertical  $\gamma_{pq}$



**Fig. 3.** A pseudo-vertical  $\gamma_{pq}$  in a pseudo-line arrangement

two paths in the dual graph of the cell complex starting at the cells above and below  $pq$ ). As  $\gamma_{pq}$  always follows a pseudo-line of  $\mathcal{A}$  or continues in a vertical direction, we note that  $\gamma_{pq}$  is  $x$ -monotone.

**Observation 1.** *The number of pseudo-lines above a point moving along  $\gamma_{pq}$  in positive  $x$ -direction is a monotone function, it increases at every crossing of  $\gamma_{pq}$  with a pseudo-line of  $\mathcal{A}$ .*

**Lemma 1.** *For any crossing  $pq$  in a pseudo-line arrangement  $\mathcal{A}$ , the curve  $\gamma_{pq}$  is a pseudo-line such that  $\mathcal{A}$  can be extended by  $\gamma_{pq}$  to a new (non-simple) pseudo-line arrangement.*

We say that pseudo-line  $a$  is above a crossing  $pq$  if  $a$  is intersected by the northbound ray of  $\gamma_{pq}$ . If  $a$  is intersected by the southbound ray of  $\gamma_{pq}$  it is considered to be below  $pq$ . (Note that this is equivalent to  $a$  separating  $pq$  from the north face or the south face, respectively.) Just like a vertical line in a line arrangement, a pseudo-vertical defines a total order on the pseudo-lines of  $\mathcal{A}$  by the order it crosses them. We denote the rank of a pseudo-line  $m \in \mathcal{A}$  in this order by  $\text{rk}_{pq}(m)$ . The following lemma shows how we can determine the rank of an element.

Let  $L(pq)$  be the set of pseudo-lines in  $\mathcal{A}$  such that each  $a \in L(pq)$  is below  $pq$  and  $a \prec p$ .

**Lemma 2.** *The northbound ray of  $\gamma_{pq}$  starting from the crossing  $pq$  until reaching an unbounded cell for the first time, follows the upper envelope of the subarrangement defined by  $L(pq) \cup \{p\}$ .*

Note that every pseudo-line that passes through the upper envelope (from below) will cross  $\gamma_{pq}$  immediately after that crossing.



**Corollary 1.** *Let  $m$  be a pseudo-line in  $\mathcal{A}$  that is above  $pq$  and for which there exists a pseudo-line  $a \in L(pq) \cup \{p\}$  such that  $a \prec m$ , i.e.,  $m$  crosses the upper envelope of  $L(pq) \cup \{p\}$  by crossing some  $e \in L(pq) \cup \{p\}$  with  $e \prec m$ . Then the rank  $\text{rk}_{pq}(m)$  equals the number of pseudo-lines above the crossing of  $e$  and  $m$ .*

If  $m$  does not intersect the upper envelope of  $L(pq) \cup \{p\}$  at some point in negative  $x$ -direction of  $pq$ , it crosses  $q$  before crossing any of the pseudo-lines of  $L(pq)$ . Therefore, we observe:

**Observation 2.** *If a pseudo-line  $m$  starts above every pseudo-line in  $L(pq)$ , then the rank of  $m$  along  $\gamma_{pq}$  is given by the number of pseudo-lines starting above  $m$  increased by 1, i.e.,  $|\{a \in \mathcal{A} : a \prec m\}| + 1$ .*

Given two different crossings  $pq$  and  $rs$  in  $\mathcal{A}$ , it is easy to see that  $\gamma_{pq}$  and  $\gamma_{rs}$  may follow the same part of a pseudo-line. Nevertheless, one can show that  $\gamma_{pq}$  and  $\gamma_{rs}$  will never intersect when drawn appropriately.

**Lemma 3.** *The set of pseudo-verticals for all crossings of a pseudo-line arrangement  $\mathcal{A}$  can be drawn such that no two pseudo-verticals intersect.*

An augmentation of  $\mathcal{A}$  with a complete collection of non-intersecting pseudo-verticals defines a total order on the vertices in the arrangement (cf. the notion of “ $P$ -augmentation” in [15]). Given an arrangement of lines, Edelsbrunner and Guibas [8, 9] define a *topological sweep* as a sweep of an arrangement of lines with a moving curve that intersects each line exactly once. The topological sweep has been generalized to pseudo-line arrangements by Snoeyink and Hershberger [27]. At any point in time during the sweep, the sweeping curve may pass over at least one crossing of the arrangement, maintaining the property that it intersects each line exactly once. However, in contrast to a straight vertical line, there can be several crossings that may be passed next by the sweep curve. It can be observed that we obtain the order of crossings determined by the pseudo-verticals by always sweeping over the lowest-possible crossing in a topological sweep.

**Lemma 4.** *The relative order of two pseudo-verticals can be obtained by a linear number of sidedness queries and queries of the form  $a \prec b$ .*

### 3 Linear-Time Pseudo-Line Selection

We now discuss algorithmic properties of pseudo-verticals. For the definition of pseudo-verticals and rank we assumed full knowledge about  $\mathcal{A}$ . The next task will be to make the notions accessible in the setting where we can only query the abstract order type through an oracle. At the end we aim at using the oracle to select a pseudo-line of given rank w.r.t. a pseudo-vertical in  $O(n)$  time.

### 3.1 An Oracle for an Arrangement

Let  $P$  be a predicate representing an abstract order type on a set  $S$  as a counterclockwise oracle, i.e., if there is a primal point set for  $S$  then  $P(x, y, z)$  tells us whether the three points form a counterclockwise oriented triangle or not, in the general setting  $P$  represents a chirotope. From [3] we borrow a linear-time procedure to determine an extreme point  $x$  of  $S$  using only queries to  $P$ . We then use the following internal representation: For all  $a \in S \setminus \{x\}$ , we define that  $x \prec a$ . For two points  $a, a' \in S \setminus \{x\}$ , we define that  $a \prec a'$  if and only if  $P(x, a, a')$ , i.e., in the arrangement the crossing  $ax$  precedes  $a'x$  on  $x$ . For two points  $p, q \in S$  with  $p \prec q$ , the dual pseudo-line  $r$  is below the crossing  $pq$  if and only if  $P(p, q, r)$ . Hence, for three points  $u, v, w \in S \setminus \{x\}$ , the dual line  $r$  is below the crossing defined by the (unordered) pair  $(u, v)$  if and only if  $P(u, v, w) = P(u, v, x)$ , i.e., above/below queries for the arrangement of pseudo-lines corresponding to  $P$  can be answered in constant time. Note that a constant number of these queries also specify whether the crossing  $ap$  precedes the crossing  $bp$  on  $p$ .

Our linear-time rank selection algorithm will depend on removing a linear fraction of the pseudo-lines in each iteration. However, the procedure must not remove the extreme point  $x$ , to keep the sub-arrangements consistent with the full arrangement.

### 3.2 Selecting a Pseudo-Line

For a given  $k$ , we want to select the pseudo-line  $m$  of rank  $k$  along  $\gamma_{pq}$ . For a subset  $B$  of pseudo-lines and  $m \in B$ , we denote with  $\text{rk}_{pq}(m, B)$  the rank of  $m$  within  $B$  on  $\gamma_{pq}$ .

In the straight-line version, a linear-time selection algorithm can be used to find an element of rank  $k$  in  $O(n)$  time. This relies on the fact that the relative position of two lines can be computed in constant time. Comparing  $\text{rk}_{pq}(s)$  and  $\text{rk}_{pq}(r)$  in the abstract setting can be reduced to deciding whether the crossing  $rs$  is below some pseudo-line  $a \in L(pq) \cup \{p\}$ . Doing this naively results in a linear number of queries and hence we get a selection algorithm with  $\Omega(n^2)$  worst-case behavior. We therefore need a more sophisticated method.

Let  $m$  be the (unknown) pseudo-line of rank  $k$  within  $B$ . We use a prune-and-search approach to identify  $m$ . By counting the elements of  $B$  above  $pq$ , we determine whether  $m$  is above or below  $pq$  (using  $O(n)$  queries). Without loss of generality, assume  $m$  is above  $pq$  (the other case is symmetric) and let  $U$  be the set of pseudo-lines above  $pq$ . Since removing pseudo-lines from  $U$  does not change the structure of the northbound part of  $\gamma_{pq}$ , we can restrict attention to  $U_B = U \cap B$ . We can also ignore (remove) pseudo-lines below  $pq$  that are not in  $L(pq)$ , i.e., each pseudo-line  $l$  below  $pq$  such that  $p \prec l$ .

As a next step, we can, in linear time, verify whether  $m$  starts above all pseudo-lines in  $L(pq) \cup \{p\}$ . If this is the case, the rank of  $m$  is determined by the order in which the pseudo-lines start, and we can apply the standard selection algorithm using this order (recall Observation 2).

We are therefore left with the case where  $m$  starts below some element  $a \in L(pq) \cup \{p\}$ . By Corollary 1, we know that we have to find the pseudo-line  $e$  where  $m$  crosses the upper envelope of  $L(pq) \cup \{p\}$  (recall that we have  $e \prec m$ ).

Basically, the algorithm continues as follows. We alternately remove elements in  $U_B$  and  $L(pq)$  such that the pseudo-lines  $e$  and  $m$  remain in the respective set until we are left with only a constant number of pseudo-lines in the arrangement. We describe these two pruning steps in two versions, first in a randomized version and after that in a deterministic version. In particular the pruning of  $U_B$  turns out to be much simpler in the randomized version.

**Randomized Pruning.** We first show how to remove pseudo-lines from  $U_B$ : Pick uniformly at random a pseudo-line  $u \in U_B$ . In time proportional to the size of  $L(pq)$  we find the last  $a \in L(pq) \cup \{p\}$  crossed by  $u$ . Since the crossing of  $u$  with  $\gamma_{pq}$  is immediately after the crossing with  $a$  we can use the crossing  $ua$  to split  $U_B$  into elements of rank less than  $\text{rk}_{pq}(u, U_B)$  and elements of larger rank. One of the two sets can be pruned.

Now we turn to removing pseudo-lines from  $L(pq)$ : One approach is to consider the  $k$ -level  $\sigma$  in the sub-arrangement induced by  $U_B$ . We observe that no element of  $L(pq)$  can cross  $\sigma$  from below before  $\sigma$  crosses the upper envelope of  $L(pq) \cup \{p\}$ , as such a pseudo-line of  $L(pq)$  would have to cross that element of  $U_B$  again before  $pq$ . All pseudo-lines in  $L(pq)$  that start below  $\sigma$  can therefore be pruned. Among the remaining elements of  $L(pq)$ , the crossings with  $\sigma$  define a total order. From the remaining elements, pick, uniformly at random, a pseudo-line  $b \in L(pq)$  and select (in  $O(n)$  time) the pseudo-line  $m' \in U_B$  where  $b$  crosses  $\sigma$ . We know that  $m'$  is unique for the choice of  $b$ . We may prune all elements  $b' \in L(pq)$  that are below  $bm'$ , as no element of  $L(pq)$  can cross  $\sigma$  more than once, and hence, no such  $b'$  can be  $e$  (the pseudo-line where  $m$  leaves the upper envelope of  $L(pq) \cup \{p\}$ ). The total order on the remaining elements in  $L(pq)$  implies that we can expect half of the elements to be pruned.

With the target of obtaining a deterministic version of our algorithm in mind, we describe the following alternative variant for pruning  $L(pq)$ . Suppose we are given any crossing  $vw$ , with  $v, w \in L(pq)$  and  $v \prec w$ , on the upper envelope of  $L(pq) \cup \{p\}$ ; see Fig. 4. Let  $\text{lv}(vw)$  be the number of pseudo-lines of  $U_B$  above the crossing  $vw$ . Depending on the value of  $\text{lv}(vw)$ , we remove the pseudo-lines of  $L(pq)$  that cannot be on the part of the upper envelope that contains the crossing with  $m$ : On  $p$  consider the crossings  $vp$  and  $wp$ . Elements of  $L(pq)$  that contribute to the upper envelope between  $vw$  and  $pq$  cross  $p$  after  $wp$ . Similarly, elements of  $L(pq)$  that contribute to the upper envelope between the north face and  $vw$  cross  $p$  before  $vp$ . Hence, depending on  $\text{lv}(vw)$ , we can remove either the pseudo-lines in  $L(pq)$  that cross  $p$  before  $wp$  or after  $vp$ . It remains to choose  $vw$  to prune enough points. The median  $t$  of the intersections of pseudo-lines from  $L(pq)$  with  $p$  can be found with a linear number of queries (even deterministically). Based on  $t$ , we partition  $L(pq)$  into the left part  $L$  and the right part  $R$ . Find the  $\prec$ -minimal element  $r^*$  of  $R$  and remove all elements  $l \in L$  with  $r^* \prec l$ . The removed elements do not contribute to the upper envelope of  $L(pq) \cup \{p\}$ . If all

elements of  $L$  are removed by that, we are done. Otherwise, we want to find the unique crossing  $vw$  of a pseudo-line  $v \in L$  and a pseudo-line  $w \in R$  on the upper envelope of  $L(pq)$ . Observe that, in the primal,  $vw$  corresponds to an edge of the convex hull of  $L(pq) \cup \{p\}$  that is stabbed by the supporting line of  $pt$ . Finding  $vw$  in linear time is described in [3]. For the sake of self-containment, we give a description of the randomized variant of that algorithm in terms of calls to our oracle. We start by picking, uniformly at random, a pseudo-line  $r \in R$  and then determine the last crossing  $rl$  with a pseudo-line from  $L$  on  $r$ . It can be argued that every pseudo-line  $r' \in R$  whose crossing with  $l$  is behind the crossing  $rl$  fails to be a candidate for  $w$  and can hence be removed (we give a proof in the full version). We expect to remove half of the pseudo-lines from  $R$  through this. A symmetric step can be used to reduce the size of  $L$ . By always applying the reduction to the larger of the two we obtain a procedure that outputs the pair  $vw$  with expected  $O(|L(pq)|)$  queries. Next we determine which elements of  $U_B$  are above and which below  $vw$ . From this we deduce whether the element  $m \in U_B$  with  $\text{rk}_{pq}(m, U_B) = k$  intersects  $\gamma_{pq}$  in the part where it follows the envelope of  $L$  or where it follows the envelope of  $R$ . Depending on this we can either prune  $L$  or  $R$  from  $L(pq)$ . The analysis of the randomized approach (given in the full version) shows that the expected number of queries is in  $O(n)$ .

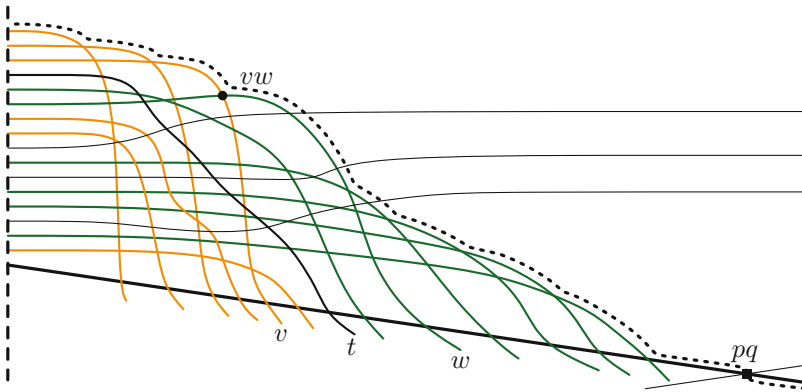


Fig. 4. Partitioning the pseudo-lines in  $L(pq)$  along  $p$  by a pseudo-line  $t$

Note that, by removing pseudo-lines from  $L(pq)$ , we obtain a new arrangement  $\mathcal{A}'$ , in which the pseudo-vertical  $\gamma_{pq}$  will, in general, follow different pseudo-lines from  $L(pq)$  along its northbound ray. Still, the number of pseudo-lines above the crossing  $em$  that we look for remains the same, and  $m$  will have the same rank with respect to the new pseudo-vertical  $\gamma'_{pq}$  in  $\mathcal{A}'$ .

**Deterministic Pruning.** To remove pseudo-lines from  $L(pq)$  deterministically, we can directly apply the deterministic algorithm given in [3] to find  $vw$  after  $O(n)$  queries.

Recall that to remove elements of  $U_B$ , we pick a pseudo-line  $u \in U_B$ . We compute the rank  $\text{rk}_{pq}(u, B)$  by finding the corresponding pseudo-line  $b \in L(pq) \cup \{p\}$  at which  $u$  passes through the upper envelope of  $L(pq) \cup \{p\}$ . Clearly, this can be done in linear time using our basic operations. If  $u = m$ , we are done. If  $\text{rk}_{pq}(u, B) < k$ , then all pseudo-lines in  $U_B$  below  $bu$  can be removed (we will see how to choose  $u$  to remove a constant fraction of the pseudo-lines). Otherwise, we remove all pseudo-lines in  $U_B$  above  $bu$  and update  $k$  accordingly. While by this operation we obtain a new arrangement  $\mathcal{A}'$ , the northbound ray of  $\gamma_{pq}$  in  $\mathcal{A}'$  is defined by the same pseudo-lines as in  $\mathcal{A}$ , and we can therefore safely continue with the next iteration. It remains to show how to pick  $u$  in a deterministic way such that at least a constant fraction of  $U_B$  can be removed. To this end, we use the concept of  $\varepsilon$ -approximation of range spaces.

Our definitions follow [20]. A *range space* is a pair  $\Sigma = (X, \mathcal{R})$  where  $X$  is a set and  $\mathcal{R}$  is a set of subsets of  $X$ . The elements of  $\mathcal{R}$  are called *ranges*. For  $X$  being finite, a subset  $A \subseteq X$  is an  $\varepsilon$ -*approximation* for  $\Sigma$  if, for every range  $R \in \mathcal{R}$ , we have  $||A \cap R|/|A| - |X \cap R|/|X|| \leq \varepsilon$ . A subset  $Y$  of  $X$  is *shattered* by  $\mathcal{R}$  if every possible subset of  $Y$  is a range of  $Y$ . The *Vapnik-Chervonenkis dimension* (VC-dimension) of  $\Sigma$  is the maximum size of a shattered subset of  $X$ . For  $|X| = n$ , the *shatter function*  $\pi_{\mathcal{R}}(n)$  of a range space  $(X, \mathcal{R})$  is defined by  $\pi_{\mathcal{R}}(n) = \max\{|\{Y \cap R : R \in \mathcal{R}\}| : Y \subseteq X\}$ . Vapnik and Chervonenkis [28] show that, for a range space  $(X, \mathcal{R})$  of VC-dimension  $d$ ,  $\pi_{\mathcal{R}}(n) \in O(n^d)$  holds. Matoušek [19, 21] gives, for a constant  $\varepsilon$ , a linear-time algorithm for computing a constant-size  $\varepsilon$ -approximation for range spaces of finite VC-dimension  $d$ , provided there exists a *subspace oracle*. A *subspace oracle* for a range space  $(X, \mathcal{R})$  is an algorithm that returns, for a given subset  $Y \subseteq X$ , the set of all distinct intersections of  $Y$  with the ranges in  $\mathcal{R}$ , i.e., the set  $\{Y \cap R : R \in \mathcal{R}\}$  and runs in time  $O(|Y| \cdot h)$ , where  $h$  is the number of sets returned. Observe that, for such a range space, the running time of the subspace oracle is bounded by  $O(|Y|^{d+1})$ , as  $h$  is at most  $\pi_{\mathcal{R}}(|Y|)$ . Let  $X$  be a point set and let  $\mathcal{R}$  consist of all possible subsets of  $X$  defined by half-planes (i.e.,  $\mathcal{R}$  is the set of *semispaces* of  $X$ ). The VC-dimension of  $(X, \mathcal{R})$  is known to be 3 [20]. We therefore have a constant-size  $\varepsilon$ -approximation for  $X$ . A subspace oracle returns, for any subset  $Y$  of points, all possible ways a line can separate  $Y$ , which can easily be done in time  $O(|Y|^3)$ . The VC-dimension of 3 for that range space holds also for abstract order types (see also [12]).

Consider again the set  $U_B$  of pseudo-lines above the crossing  $pq$  in  $\mathcal{A}$ . We obtain an  $\varepsilon$ -approximation  $A \subset U$  for the range space of semispaces.  $A$  is of constant size for a fixed  $\varepsilon$ . For each pseudo-line  $o \in A$ , we obtain the crossing of  $o$  with the pseudo-lines in  $L(pq)$  that defines the rank  $\text{rk}_{pq}(o)$  in  $O(n)$  time. Let  $u_A \in A$  have the median rank among the elements of  $A$ . Then not less than  $1/2 - \varepsilon$  pseudo-lines of  $U$  are above and below the crossing  $bu_A$  on the upper envelope of  $L(pq)$ ; we may prune a constant fraction of the elements in  $U_B$ .

**Analysis.** In each iteration we prune the larger of  $U$  and  $L(pq)$ . In both cases, we remove at least half of the pseudo-lines on one side of  $pq$ , and therefore

$n/4 - O(1)$  pseudo-lines in each iteration. Since each iteration takes  $O(n)$  time, we have overall a linear-time prune-and-search algorithm.

**Theorem 1.** *Given an arrangement  $\mathcal{A}$  of pseudo-lines, a subset  $B$  of its pseudo-lines, a crossing  $pq$ , and a natural number  $k \leq |B|$ , the pseudo-line  $m \in B$  of rank  $k$  in  $B$  on the pseudo-vertical through  $pq$ , i.e.,  $\text{rk}_{pq}(m, B)$ , can be found in linear time using only sidedness queries.*

A detailed analysis of the LMS algorithm (given in the full version) reveals that selecting an intersection point at a vertical line is the only part of the algorithm that cannot directly be implemented using only sidedness queries.

Compared to  $L(pq)$ , pruning  $U_B$  deterministically required the technically involved step of selecting an  $\epsilon$ -approximation. Is there a more “light-weight” deterministic way to prune  $U_B$ , similar to the one used for  $L(pq)$ ?

## References

1. Aichholzer, O., Hackl, T., Korman, M., Pilz, A., Vogtenhuber, B.: Geodesic-Preserving Polygon Simplification. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) Algorithms and Computation. LNCS, vol. 8283, pp. 11–21. Springer, Heidelberg (2013)
2. Aichholzer, O., Korman, M., Pilz, A., Vogtenhuber, B.: Geodesic Order Types. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 216–227. Springer, Heidelberg (2012)
3. Aichholzer, O., Miltzow, T., Pilz, A.: Extreme point and halving edge search in abstract order types. *Comput. Geom.* **46**(8), 970–978 (2013)
4. Avnaim, F., Boissonnat, J.D., Devillers, O., Preparata, F.P., Yvinec, M.: Evaluating signs of determinants using single-precision arithmetic. *Algorithmica* **17**(2), 111–132 (1997)
5. Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. *J. Comput. System Sci.* **7**(4), 448–461 (1973)
6. Boissonnat, J.D., Snoeyink, J.: Efficient algorithms for line and curve segment intersection using restricted predicates. *Comput. Geom.* **16**(1), 35–52 (2000)
7. Bose, P., Demaine, E.D., Hurtado, F., Iacono, J., Langerman, S., Morin, P.: Geodesic ham-sandwich cuts. In: SoCG. pp. 1–9. ACM (2004)
8. Edelsbrunner, H., Guibas, L.J.: Topologically sweeping an arrangement. *J. Comput. Syst. Sci.* **38**(1), 165–194 (1989)
9. Edelsbrunner, H., Guibas, L.J.: Corrigendum: Topologically sweeping an arrangement. *J. Comput. Syst. Sci.* **42**(2), 249–251 (1991)
10. Edelsbrunner, H., Waupotitsch, R.: Computing a ham-sandwich cut in two dimensions. *J. Symb. Comput.* **2**(2), 171–178 (1986)
11. Erickson, J.G.: Lower Bounds for Fundamental Geometric Problems. Ph.D. thesis, University of California at Berkeley (1996)
12. Gärtner, B., Welzl, E.: Vapnik-Chervonenkis dimension and (pseudo-)hyperplane arrangements. *Discrete Comput. Geom.* **12**, 399–432 (1994)
13. Goodman, J.E., Pollack, R.: On the combinatorial classification of nondegenerate configurations in the plane. *J. Comb. Theory, Ser. A* **29**(2), 220–235 (1980)
14. Goodman, J.E., Pollack, R.: Multidimensional sorting. *SIAM J. Comput.* **12**(3), 484–507 (1983)

15. Goodman, J.E., Pollack, R.: Semispaces of configurations, cell complexes of arrangements. *J. Combin. Theory Ser. A* **37**(3), 257–293 (1984)
16. Goodman, J.E., Pollack, R.: Allowable sequences and order types in discrete and computational geometry. In: Pach, J. (ed.) *New Trends in Discrete and Computational Geometry*, pp. 103–134. Springer (1993).
17. Knuth, D.E.: *Axioms and Hulls*, vol. 606, LNCS. Springer (1992)
18. Lo, C.Y., Matoušek, J., Steiger, W.: Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.* **11**, 433–452 (1994)
19. Matoušek, J.: Approximations and optimal geometric divide-and-conquer. In: *STOC*, pp. 505–511. ACM (1991)
20. Matoušek, J.: Epsilon-nets and computational geometry. In: Pach, J. (ed.) *New Trends in Discrete and Computational Geometry*, pp. 69–89. Springer (1993)
21. Matoušek, J.: Approximations and optimal geometric divide-and-conquer. *J. Comput. System Sci.* **50**(2), 203–208 (1995)
22. Megiddo, N.: Partitioning with two lines in the plane. *J. Algorithms* **6**(3), 430–433 (1985)
23. Meikle, L.I., Fleuriot, J.D.: Mechanical Theorem Proving in Computational Geometry. In: Hong, H., Wang, D. (eds.) *ADG 2004*. LNCS (LNAI), vol. 3763, pp. 1–18. Springer, Heidelberg (2006)
24. Pichardie, D., Bertot, Y.: Formalizing Convex Hull Algorithms. In: Boulton, R.J., Jackson, P.B. (eds.) *TPHOLs 2001*. LNCS, vol. 2152, pp. 346–361. Springer, Heidelberg (2001)
25. Pilz, A.: *On the Complexity of Problems on Order Types and Geometric Graphs*. Ph.D. thesis, Graz University of Technology (2014)
26. Roy, S., Steiger, W.: Some combinatorial and algorithmic applications of the Borsuk-Ulam theorem. *Graphs Combin.* **23**(1), 331–341 (2007)
27. Snoeyink, J., Hershberger, J.: Sweeping arrangements of curves. In: *SoCG*, pp. 354–363 (1989)
28. Vapnik, V.N., Chervonenkis, A.Ya.: On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.* **16**, 264–280 (1971)

# **Network and Scheduling Algorithms**



# Graph Orientation and Flows over Time

Ashwin Arulsevan<sup>1</sup>, Martin Groß<sup>2</sup>(✉), and Martin Skutella<sup>2</sup>

<sup>1</sup> Department of Management Science, University of Strathclyde, Glasgow, UK

`ashwin.arulsevan@strath.ac.uk`

<sup>2</sup> Institut für Mathematik, TU Berlin, Str. des 17. Juni 136, 10623 Berlin, Germany

`{gross,skutella}@math.tu-berlin.de`

**Abstract.** Flows over time are used to model many real-world logistic and routing problems. The networks underlying such problems – streets, tracks, etc. – are inherently undirected and directions are only imposed on them to reduce the danger of colliding vehicles and similar problems. Thus the question arises, what influence the orientation of the network has on the network flow over time problem that is being solved on the oriented network. In the literature, this is also referred to as the *contraflow* or *lane reversal* problem.

We introduce and analyze the *price of orientation*: How much flow is lost in any orientation of the network if the time horizon remains fixed? We prove that there is always an orientation where we can still send  $\frac{1}{3}$  of the flow and this bound is tight. For the special case of networks with a single source or sink, this fraction is  $\frac{1}{2}$  which is again tight. We present more results of similar flavor and also show non-approximability results for finding the best orientation for single and multicommodity maximum flows over time.

## 1 Introduction

Robbins [13] studied the problem of orienting streets as early as 1939, motivated by the problem of controlling congestion by making streets of a city one-way during the weekend. He showed that a strongly connected digraph could be obtained by orienting the edges of an undirected graph if and only if it is 2-edge connected.

The problem of prescribing or changing the direction of road lanes is a strategy employed to mitigate congestion during an emergency situation or at rush hour. This is called a *contraflow* problem (or sometimes *reversible flow* or *lane reversal* problem). Contraflows are an important tool for hurricane evacuation [18], and

---

Supported by the DFG Priority Program Algorithms for Big Data (SPP 1736) and by the DFG Research Center MATHEON “Mathematics for key technologies” in Berlin. See also <http://arxiv.org/abs/1409.3081> for a version of this paper with all proofs.

Ashwin Arulsevan: The work was performed while the author was working at TU Berlin.

in that context the importance of modeling time has become prevalent in the past decade [19]. It is also employed to handle traffic during rush hours [6].

*Flows over time* (also referred to as *dynamic flows*) have been introduced by Ford and Fulkerson [4] and extend the classic notion of static network flows. They can model a time aspect and are therefore better suited to represent real-world phenomena such as traffic, production flows or evacuations. For the latter, *quickest* flows (over time) are the model of choice. They are based on the idea that a given number of individuals should leave a dangerous area as quickly as possible [1, 3]. Such an evacuation scenario is modeled by a network, with nodes representing locations. Nodes containing evacuees are denoted as *sources* while the network's *sinks* model safe areas or exits. For networks with multiple sources and sinks, quickest flows are also referred to as *quickest transshipments* [9] and the problem of computing them is called an *evacuation problem* [15]. A strongly polynomial algorithm for the quickest flow problem was described in [8]. For a more extensive introduction to flows over time, see [14].

In this paper, we are interested in combining the orientation of a network with flows over time – we want to orient the network such that the orientation is as beneficial as possible for the flow over time problem. We will assume that we can orient edges in the beginning, and cannot change the orientation afterwards. The assumption is reasonable in an evacuation setting as altering the orientation in the middle of an evacuation process can be difficult or even infeasible, depending on the resources available. We also assume that each edge has to be routed completely in one direction – but this will not impose any restriction to our modeling abilities, as we can model lanes with parallel edges if we want to orient them individually.

If there is only a single source and sink, we can apply the algorithm of Ford and Fulkerson [4] to obtain an orientation and a solution. Furthermore, it was shown that finding the best orientation for a quickest flow problem with multiple sources and sinks is NP-hard [10, 12]. Due to the hardness of the problem, heuristic and simulation tools are predominantly used in practice [10, 16–18].

*Our Contribution.* In Section 3 we study the *price of orientation* for networks with single and multiple sources and sinks, i. e., we deal with the following questions: How much flow is lost in any orientation of the network given a fixed time horizon? And how much longer do we need in any orientation to satisfy all supplies and demands, compared to the undirected network?

The price of orientation can also be seen as a comparison tool for different orientation models (e.g., models that allow to change orientations multiple times vs. models which fix the orientations at the beginning). We will study the price of orientation for the case where the orientation has to be fixed at the beginning, as this model is the easiest to realize for applications like evacuations, and leave other models to further research.

To our knowledge, the price of orientation has not been studied for flows over time so far. It follows from the work of Ford and Fulkerson [4] that for *s-t*-flows over time the price of orientation is 1: Ford and Fulkerson proved that a maximum flow over time can be obtained by temporally repeating a static

**Table 1.** An overview of price of orientation results

Sources	Sinks	Flow Value		Time	
		Price	Reference	Price	Reference
1	1	1	Ford, Fulkerson [4]	1	Ford, Fulkerson [4]
2+	1	2	Theorem 3	$\Omega(n)$	Theorem 4
1	2+	2	Theorem 3	$\Omega(n)$	Theorem 4
2+	2+	3	Theorem 1, 2	$\Omega(n)$	Theorem 4

min-cost flow and thus uses every edge in one direction only. The latter property no longer holds if there is more than one source or sink.

We are able to give tight bounds for the price of orientation with regard to the flow value, and we show that the price of orientation with regard to the time horizon cannot be smaller than linear in the number of nodes. Table 1 shows an overview of our results. Our main result is the tight bound of 3 on the flow price of orientation for the multiple sources and sinks case. We describe an algorithm that is capable of simulating balances through capacities of auxiliary edges. This allows us to transform a problem with supplies and demands to the much simpler case of a single source with unbounded supply and a single sink with unbounded demand. We characterize the properties that the capacities of the auxiliary edges should have for a good approximation, and describe how they can be obtained using an iterative approach that uses Brouwer fixed-points. On the negative side, we give an instance whose price of orientation is not better than 3.

Since we have two ways to pay the price of orientation – decreasing the flow value or increasing the time horizon – the question arises whether it might be desirable to pay the price partly as flow value and partly as time horizon. We prove that by doing so, we can achieve a bicriteria-price of 2/2 for the case of multiple sources and sinks, i. e., we can send at least half the flow value in twice the amount of time.

In Section 4 we analyze the complexity of finding the best orientation to minimize the loss in time or flow value for a specific instance. We are able to show that these problems cannot be approximated with a factor better than 2, unless  $P = NP$ . Furthermore, we extend this to two multicommodity versions of this problem and show that these become inapproximable, unless  $P = NP$ .

## 2 Preliminaries

*Networks and Orientations.* An undirected network over time  $N$  consists of an undirected graph  $G$  with a set of nodes  $V(G)$ , a set of edges  $E(G)$ , capacities  $u_e \geq 0$  and transit times  $\tau_e \geq 0$  on all edges  $e \in E(G)$ , balances  $b_v$  on all nodes  $v \in V(G)$ , and a time horizon  $T \geq 0$ . For convenience, we define  $V(N) := V(G)$ ,  $E(N) := E(G)$ . The capacity  $u_e$  is interpreted as the maximal inflow rate of edge  $e$  and flow entering an edge  $e$  with a transit time of  $\tau_e$  at time  $\theta$  leaves

$e$  at time  $\theta + \tau_e$ . We extend the edge and node attributes to sets of edges and nodes by defining:  $u(E) := \sum_{e \in E} u_e$ ,  $\tau(E) := \sum_{e \in E} \tau_e$  and  $b(V) := \sum_{v \in V} b_v$ . We denote the set of edges incident to a node  $v$  by  $\delta(v)$ .

We define  $S^+ := \{v \in V(G) \mid b_v > 0\}$  as the set of nodes with positive balance (also called *supply*), which we will refer to as *sources*. Likewise, we define  $S^- := \{v \in V(G) \mid b_v < 0\}$  as the set of nodes with negative balance (called *demand*), which we will refer to as *sinks*. Additionally, we assume that  $\sum_{v \in V(G)} b_v = 0$  and define  $B := \sum_{v \in S^+} b_v$ . To define a *directed* network over time, replace the undirected graph with a directed one. In a directed network, we denote the set of edges leaving a node  $v$  by  $\delta^+(v)$  and the set of edges entering  $v$  by  $\delta^-(v)$  for all  $v \in V(G)$ .

An *orientation*  $\vec{N}$  of an undirected network over time  $N$  is a *directed network over time*  $\vec{N} = (\vec{G}, \vec{u}, b, \vec{\tau}, T)$ , such that  $\vec{G}$ ,  $\vec{u}$  and  $\vec{\tau}$  are orientations of  $G$ ,  $u$  and  $\tau$ , respectively. This means that for every edge  $\{v, w\} \in E(G)$  there is either  $(v, w)$  or  $(w, v)$  in  $E(\vec{G})$  (but not both) and (assuming  $(v, w) \in E(\vec{G})$ )  $\vec{u}_{(v,w)} = u_{\{v,w\}}$  and  $\vec{\tau}_{(v,w)} = \tau_{\{v,w\}}$ . Recall that we can use parallel edges if we want to model streets with multiple lanes – each parallel edge can then be oriented individually.

*Flows over Time.* A *flow over time*  $f$  in a directed network over time  $N = (G, u, b, \tau, T)$  assigns a Lebesgue-integrable flow rate function  $f_e : [0, T) \rightarrow \mathbb{R}_0^+$  to every edge  $e \in E(G)$ . We assume that no flow is left on the edges after the time horizon, i. e.,  $f_e(\theta) = 0$  for all  $\theta \geq T - \tau_e$ . The flow rate functions  $f_e$  have to obey *capacity constraints*, i. e.,  $f_e(\theta) \leq u_e$  for all  $e \in E, \theta \in [0, T)$ . Furthermore, they have to satisfy *flow conservation constraints*. For brevity, we define the *excess* of a node as the difference between the flow reaching the node and leaving it:  $\text{ex}_f(v, \theta) := \sum_{e \in \delta^-(v)} \int_0^{\theta - \tau_e} f_e(\xi) d\xi - \sum_{e \in \delta^+(v)} \int_0^\theta f_e(\xi) d\xi$ . Additionally, we define  $\text{ex}(v) := \text{ex}(v, T)$ . Then we can write the flow conservation constraints as

$$\begin{aligned} \text{ex}(v) = 0, \text{ex}(v, \theta) \geq 0 & \quad \text{for all } v \in V(N) \setminus (S^+ \cup S^-), \theta \in [0, T), \\ 0 \geq \text{ex}(v, \theta) \geq -b_v & \quad \text{for all } v \in S^+, \theta \in [0, T), \\ 0 \leq \text{ex}(v, \theta) \leq -b_v & \quad \text{for all } v \in S^-, \theta \in [0, T). \end{aligned}$$

The *value*  $|f|_\theta$  of a flow over time  $f$  until time  $\theta$  is the amount of flow that has reached the sinks until time  $\theta$ :  $|f|_\theta := \sum_{s^- \in S^-} \text{ex}_f(s^-, \theta)$  with  $\theta \in [0, T]$ . For brevity, we define  $|f| := |f|_T$ .

We define flows over time in undirected networks over time  $N$  by transforming  $N$  into a directed network  $N'$ , using the following construction. We replace every undirected edge  $e = \{v, w\} \in E(N)$  by introducing two additional nodes  $vw, vw'$  and edges  $(v, vw), (w, vw), (vw, vw'), (vw', v), (vw', w)$ . We set  $u_{(vw, vw')} = u_e$  and  $\tau_{(vw, vw')} = \tau_e$ , the rest of the new edges gets zero transit times and infinite capacities. This transformation replaces all undirected edges with directed edges, giving us the directed network  $N'$ . Every flow unit that could have used  $\{v, w\}$  from either  $v$  to  $w$  or  $w$  to  $v$  must now use the new edge  $(vw, vw')$ , which has the same attributes as  $\{v, w\}$ . The other four edges just ensure that  $(vw, vw')$

can be used by flow from  $v$  to  $w$  or  $w$  to  $v$ . Thus, whenever we consider flows over time in  $N$ , we interpret them as flows over time in  $N'$  instead.

*Maximum Flows over Time.* The *maximum flow over time problem* consists of a directed or undirected network over time  $N = (G, u, b, \tau, T)$  where the objective is to find a flow over time of maximum value. The sources and sinks have usually unbounded supplies and demands in this setting but it can also be studied with finite supplies and demands. In the latter case, the problem is sometimes referred to as *transshipment over time* problem.

Ford and Fulkerson [4] showed that the case of unbounded supplies and demands can be solved by a reduction to a static minimum cost circulation problem. This yields a *temporally repeated* flow as an optimal solution. Such a flow  $x$  is given by a family of paths  $\mathcal{P}$  along which flow is sent at constant rates  $x_P$ ,  $P \in \mathcal{P}$  during the time intervals  $[0, T - \tau_P)$ , with  $\tau_P := \sum_{e \in P} \tau_e$ . The algorithm of Ford and Fulkerson obtains these paths by decomposing the solution to the minimum cost circulation problem. This algorithm has the nice property that edges are only used in one direction, as it is based on a static flow decomposition.

The *maximum contraflow over time problem* is given by an undirected network over time  $N = (G, u, b, \tau, T)$  and the objective is to find an orientation  $\vec{N}$  of  $N$  such that the value of a maximum flow over time in  $\vec{N}$  is maximal over all possible orientations of  $N$ .

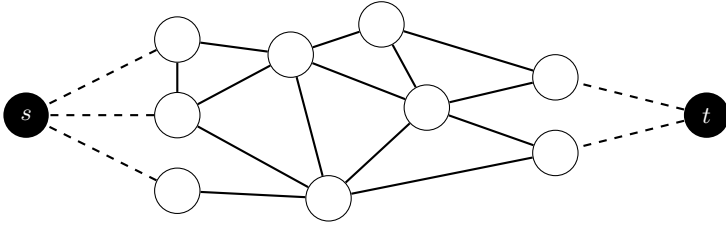
*Quickest Flows.* The *quickest flow problem* or *quickest transshipment problem* is given by a directed or undirected network over time  $N = (G, u, b, \tau)$  and the objective is to find the smallest time horizon  $T$  such that all supplies and demands can be fulfilled, i. e., a flow over time with value  $B$  can be sent. Hoppe and Tardos [8] gave a polynomial algorithm to solve this problem. However, an optimal solution to this problem might have to use an edge in both directions. The *quickest contraflow problem* is given by an undirected network over time  $N = (G, u, b, \tau)$  and the objective is to find an orientation  $\vec{N}$  of  $N$  such that the time horizon of a quickest flow in  $\vec{N}$  is minimal over all possible orientations of  $N$ .

### 3 The Price of Orientation

We study two different models for the price of orientation. The *flow price of orientation* for an undirected network over time  $N = (G, u, b, \tau, T)$  is the ratio between the value of a maximum flow over time  $f_N$  in  $N$  and maximum of the values of maximum flows over time  $f_{\vec{N}}$  in orientations  $\vec{N}$  of  $N$ :

$$|f_N| / \max_{\vec{N} \text{ orientation of } N} |f_{\vec{N}}|.$$

Similarly, the *time price of orientation* for an undirected network over time  $N = (G, u, b, \tau)$  is the ratio between the minimal time horizon  $T(f_{\vec{N}})$  of a quickest



**Fig. 1.** The modified network consisting of the original network (white), the superterminals (black) and the dashed auxiliary edges

flow  $f_{\vec{N}}$  in an orientation  $\vec{N}$  of  $N$  and the time horizon  $T(f_N)$  of a quickest flow over time  $f_N$  in  $N$ :

$$\min_{\vec{N} \text{ orientation of } N} T(f_{\vec{N}})/T(f_N).$$

### 3.1 Price in Terms of Flow Value

In this subsection, we will examine the flow price of orientation. We will see that orientation can cost us two thirds of the flow value in some instances, but not more.

**Theorem 1.** *Let  $N = (G, u, b, \tau, T)$  be an undirected network over time, in which  $B$  units of flow can be sent within the time horizon  $T$ . Then there exists an orientation  $\vec{N}$  of  $N$  in which at least  $B/3$  units of flow can be sent within time horizon  $T$ .*

*Proof.* The idea of this proof is to simplify the instance, such that a temporally repeated solution can be found. Such a solution gives us an orientation that we can use, if the simplification does not cost us too much in terms of flow value. We will achieve this by simulating the balances using additional edges and capacities, creating a maximum flow over time problem which permits a temporally repeated solution. Then we show that the resulting maximum flow over time problem is close enough to the original problem for our claim to follow.

*Simulating the Balances.* We achieve this by adding a super source  $s$  and a super sink  $t$  to the network, resulting in an undirected network over time  $N' = (G', u', \tau', s, t, T)$  with  $V(G') := V(G) \cup \{s, t\}$ ,  $E(G') := E(G) \cup \{\{s, s^+\}\} \cup \{s^+ \in S^+\} \cup \{\{s^-, t\} \mid s^- \in S^-\}$ ,  $u'_e := u_e$  for  $e \in E(G)$  and  $\infty$  otherwise,  $\tau'_e := \tau_e$  for  $e \in E(G)$  and 0 otherwise. We refer to the newly introduced edges of  $E(G') \setminus E(G)$  as *auxiliary* edges. Furthermore, we sometimes refer to an auxiliary edge by the unique terminal node it is adjacent to and write  $u_v$  for  $u_e, e = (s, v)$ ,  $f_v$  for  $f_e, e = (s, v)$  and so on. An illustration of this construction can be found in Fig. 1.

The network  $N'$  describes a maximum flow over time problem which has an optimal solution that is a temporally repeated flow, which uses each edge only in

one direction during the whole time interval  $[0, T]$ . Thus, there is an orientation  $\vec{N}'$  such that the value of a maximum flow over time in  $N'$  is the same as in  $N'$ . However, an optimal solution for  $N'$  will generally be infeasible for  $N$ , since there are no balances in  $N'$ .

Thus, we need to modify  $N'$  such that balances of  $N$  are respected – but without using actual balances. This leaves us the option to modify the capacities of the auxiliary edges. In the next step, we will show that we can always find capacities that enforce that the balances constraints are satisfied and have nice properties for bounding the loss in flow value incurred by the capacity modification. These properties are then used in the last step to complete the proof.

*Enforcing Balances by Capacities for Auxiliary Edges.* In this step, we show that we can choose capacities for the auxiliary edges in such a way that there is a maximum flow over time in the resulting network that respects the original balances. Choosing finite capacities for some of the auxiliary edges will – in general – reduce the maximum flow value that can be sent, though. In order to bound this loss of flow later on, we need capacities with nice properties, that can always be found.

**Lemma 1.** *There are capacities  $u''_e$  that differ from  $u'_e$  only for the auxiliary edges, such that the network  $N'' = (G', u'', \tau', s, t, T)$  has a temporally repeated maximum flow over time  $f$  with the following properties*

- *the balances of the nodes in the original setting are respected:*  
 $|f_v| := \int_0^T f_v(\theta) d\theta \leq |b_v| \quad \forall v \in S^+ \cup S^-,$
- *and that terminals without tightly fulfilled balances have auxiliary edges with unbounded capacity:  $|f_v| < |b_v| \Rightarrow u_v = \infty \quad \forall v \in S^+ \cup S^-.$*

*Proof.* The idea of this proof is to start with unbounded capacities and iteratively modify the capacities based on the balance and amount of flow currently going through an node, until we have capacities satisfying our needs. In order to show that such capacities exist, we apply Brouwer’s fixed-point theorem on the modification function to show the existence of a fix point. By construction of the modification function, this implies the existence of the capacities.

*Prerequisites for Using Brouwer’s Fixed Point-Theorem.* We begin by defining  $U := \sum_{v \in S^+} \sum_{a=(v, \cdot) \in E(G)} u_a$  as an upper bound for the capacity of auxiliary edges and we will treat  $U$  and  $\infty$  interchangeably from now on. This allows us to consider capacities in the interval  $[0, U]$ , which is convex and compact, instead of  $[0, \infty)$ . This will be necessary for applying Brouwer’s fixed point theorem later on.

Now assume that we have some capacities  $u \in [0, U]^{S^+ \cup S^-}$  for the auxiliary edges. Since we leave the capacities for all other edges unchanged, we identify the capacities for the auxiliary edges with the capacities for all edges. Compute a maximum flow over time  $f(u)$  for  $(G', u, \tau', s, t, T)$  by using Ford and Fulkerson’s reduction to a static minimum cost flow. For this proof, we need to ensure that small changes in  $u$  result in small changes in  $f(u)$ , i. e., we need continuity. Thus,

we will now specify that we compute the minimum cost flow by using successive computations of shortest  $s$ - $t$ -paths. In case there are multiple shortest paths in an iteration, we consider the shortest path graph, and choose a path in this graph by using a depth-first-search that uses the order of edges in the adjacency list of the graph as a tie-breaker. The path decomposition of the minimum cost flow deletes paths in the same way. This guarantees us that we choose paths consistently, leading to the continuity that we need.

*Defining the Modification Function.* In order to obtain capacities for a maximum flow over time that respects the balances, we define a function  $h : [0, U]^{S^+ \cup S^-} \rightarrow [0, U]^{S^+ \cup S^-}$  which will reduce the capacities of the auxiliary edges, if balances are not respected:

$$(h(u))_v := \min \left\{ U, \frac{b_v}{|f_v(u)|} u_v \right\} \quad \forall v \in S^+ \cup S^-.$$

$|f_v(u)|$  refers to the amount of flow going through the auxiliary edge of terminal  $v \in S^+ \cup S^-$  in this definition. If  $|f_v(u)| = 0$ , we assume that the minimum is  $U$ . Due to our rigid specification in the maximum flow computation,  $|f_v(u)|$  is continuous, and therefore  $h$  is continuous as well.

*Using Brouwer’s Fixed-Point Theorem.* Thus,  $h$  is continuous over a convex, compact subset of  $\mathbb{R}^{S^+ \cup S^-}$ . By Brouwer’s fixed-point theorem it has a fixed point  $\bar{u}$  with  $h(\bar{u}) = \bar{u}$ , meaning that for every  $v \in S^+ \cup S^-$  either  $u_v = U$  or  $u_v = \frac{b_v}{|f_v(u)|} u_v \Leftrightarrow b_v = |f_v(u)|$  holds, which is exactly what we require of our capacities. □

We can now choose capacities  $u''$  in accordance to Lemma 1, and thereby gain a maximum flow over time problem instance  $N'' = (G', u'', \tau', s, t, T)$ , that has a temporally repeated optimal solution which does not violate the original balances. What is left to do is to analyze by how much the values of optimal solutions for  $N$  and  $N''$  are apart.

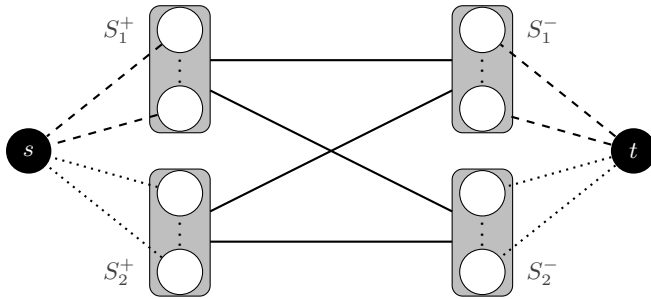
*Bounding the Difference in Flow Value between  $N$  and  $N''$ .* We now want to show that we can send at least  $B/3$  flow units in the network  $N''$  with the auxiliary capacities of the previous step. For the purpose of this analysis, we partition the sources and sinks as follows.

$$\begin{aligned} S_1^+ &:= \{s^+ \in S^+ \mid u_{s^+} < \infty\}, S_2^+ := \{s^+ \in S^+ \mid u_{s^+} = \infty\}, \\ S_1^- &:= \{s^- \in S^- \mid u_{s^-} < \infty\}, S_2^- := \{s^- \in S^- \mid u_{s^-} = \infty\}. \end{aligned}$$

The partitioning is also shown in Fig. 2.

Now let  $f$  be a temporally repeated maximum flow in  $N''$  that does not violate balances. Notice that the auxiliary edges to terminals in  $S_2^+$  and  $S_2^-$ , respectively, have infinite capacity and that the supply / demand of nodes in  $S_1^+$  and  $S_1^-$  is fully utilized. Thus,  $|f| \geq \max \{b(S_1^+), b(S_1^-)\}$ . Should  $b(S_1^+) \geq B/3$  or  $b(S_1^-) \geq B/3$  hold, we would be done – so let us assume that  $b(S_1^+) < B/3$  and  $b(S_1^-) < B/3$ . It follows that  $b(S_2^+) \geq 2/3B$  and  $b(S_2^-) \geq 2/3B$  must





**Fig. 2.** The partitioning based on the capacities of the auxiliary edges. Dashed edges have finite capacity, dotted edges have infinite capacities.

hold in this case. Now consider the network  $N'$  with the terminals of  $S_1^+$  and  $S_1^-$  removed, leaving only the terminals of  $S_2^+$  and  $S_2^-$ . We call this network  $N'(S_2^+, S_2^-)$ . Let  $|f'|$  be the value of a maximum flow over time in  $N'(S_2^+, S_2^-)$ . Since  $B$  units of flow can be sent in  $N$  (and therefore  $N'$  as well), we must be able to send at least  $B/3$  units in  $N'(S_2^+, S_2^-)$ . This is due to the fact that  $b(S_2^+) \geq 2/3B, b(S_2^-) \geq 2/3B$  – even if  $B/3$  of these supplies and demands were going to  $S_1^-$  and coming from  $S_1^+$ , respectively, this leaves at least  $B/3$  units that must be sent from  $S_2^+$  to  $S_2^-$ . Thus,  $B/3 \leq |f'|$ . Since the capacities of the auxiliary edges of  $S_2^+$  and  $S_2^-$  are infinite, we can send these  $B/3$  flow units in  $N''$  as well, proving this part of the claim.

Thus, we have shown that a transshipment over time problem can be transformed into a maximum flow over time problem with auxiliary edges and capacities. If these edges and capacities fulfill the requirements of Lemma 1, we can transfer solutions for the maximum flow problem to the transshipment problem such that at least one third of the total supplies of the transshipment problem can be sent in the flow problem. Finally, the proof of Lemma 1 shows that such capacities do always exist, completing the proof.  $\square$

Notice that the algorithm described in the proof is not efficient – it relies on Brouwer’s fixed-point theorem, and finding an (approximate) Brouwer fixed-point is known to be PPAD-complete [11] and exponential lower bounds for the common classes of algorithms for this problem are known [7]. Since the algorithm is efficient aside from finding a Brouwer fixed-point, our problem is at least not harder than finding a Brouwer fixed-point. Thus, our problem is probably not FNP-complete (with FNP being the functional analog of NP) as PPAD-completeness indicates that a problem is not FNP-complete [11]. However, it is possible that the fixed-point can efficiently be found for the specific function we are interested in. One problem for finding such an algorithm is however, that changing the capacity of one auxiliary edge does not only modify the amount flow through its associated terminal but through other terminals as well – and this change in flow value can be an increase or decrease, making monotonicity arguments problematic.

Another potential approach could be to find a modification function for which (approximate) Brouwer fixed-points can be found efficiently. Using approximate Brouwer fixed-points would result in a weaker version of Lemma 1, where an additional error is introduced due to the approximation. This error can be made arbitrarily small by approximating the Brouwer fixed-point more closely, or by using alternative modification functions. However, finding a modification function for which an approximation of sufficient quality can be found efficiently remains an open question.

Now that we have an upper bound for the flow price of orientation and it turns out that this bound is tight. The proof for the next theorem can be found in the full version of the paper.

**Theorem 2.** *For any  $\varepsilon > 0$ , there are undirected networks over time  $N = (G, u, b, \tau, T)$  in which  $B$  units of flow can be sent, but at most  $B/3 + \varepsilon$  units of flow can be sent in any orientation  $\vec{N}$  of  $N$ .*

With these theorems, we have a tight bound for the flow price of orientation in networks with arbitrarily many sources and sinks. In the case of a single source and sink, we have a maximum flow over time problem and we can always find an orientation in which we can send as much flow as in the undirected network. This leaves the question about networks with either a single source or a single sink open. However, if we use the knowledge that only one source (or sink) exists in the analysis done in the proofs of Theorem 1 and Theorem 2, we achieve a tight factor of 2 in these cases (the proof can again be found in the full version of the paper).

**Theorem 3.** *Let  $N = (G, u, b, \tau, T)$  be an undirected network over time with a single source or sink, in which  $B$  units of flow can be sent within the time horizon  $T$ . Then there exists an orientation  $\vec{N}$  of  $N$  in which at least  $B/2$  units of flow can be sent within time horizon  $T$ , and there are undirected networks over time for which this bound is tight.*

### 3.2 Price in Terms of the Time Horizon

In this part, we examine by how much we need to extend the time horizon in order to send as much flow in an orientation as in the undirected network. It turns out that there are instances for which we have to increase the time horizon by a factor that is linear in the number of nodes. This is due to the fact that we have to send everything, which can force us to send some flow along very long detours – this is similar to what occurs in [5]. For this reason it is not a good idea to pay the price of orientation in time alone. A detailed proof of the theorem can be found in the full version of the paper.

**Theorem 4.** *There are undirected networks over time  $N = (G, u, b, \tau, T + 1)$  with either a single source or a single sink in which  $B$  units of flow can be sent within a time horizon of  $T$ , but it takes a time horizon of at least  $(n - 1)/4 \cdot T$  to send  $B$  units of flow in any orientation  $\vec{N}$  of  $N$ . This bound also holds if  $G$  is a tree with multiple sources and sinks.*

A similar bound holds for trees with unit-capacities – details can be found in the full version of the paper.

### 3.3 Price in Terms of Flow and Time Horizon

We have seen now that the price of orientation is 3 with regard to the flow value, and  $\Omega(n)$  with regard to the time horizon. We can improve on these bounds if we allow to pay the price of orientation partly in terms of flow value and partly in terms of the time horizon. This is possible by combining the reduction to maximum flows over time from Theorem 1 with the concept of temporally averaged flows (see , e. g., [2]). The proof can be found in the full version of the paper.

**Theorem 5.** *Let  $N = (G, u, b, \tau, T)$  be an undirected network over time, in which  $B$  units of flow can be sent within the time horizon  $T$ . Then there exists an orientation  $\vec{N}$  of  $N$  in which at least  $B/2$  units of flow can be sent within time horizon  $2T$ . The orientation and a transshipment over time with this property can be obtained in polynomial time.*

**Earliest Arrival Flows.** We now have tight bounds for the flow and time price of orientation for maximum or quickest flows over time. However, for application in evacuations, it would be nice if we could analyze the price of orientation for so-called earliest arrival flows as well, as they provide guarantees for flow being sent at all points in time. Unfortunately, we can create instances where not even approximate earliest arrival contraflows exist, because the trade-off between different orientations becomes too high. Details can be found in full version of the paper.

## 4 Complexity Results

Furthermore, we can show non-approximability results for several contraflow over time problems. More specifically, we can show that neither quickest contraflows nor maximum contraflows over time can be approximated better than a factor of 2, unless  $P = NP$ . For multicommodity contraflows over time, we can even show that maximum multicommodity concurrent contraflows and quickest multicommodity contraflows cannot be approximated at all, even with zero transit times, unless  $P = NP$ . The complete theorems and proofs can be found in the full version of the paper.

**Acknowledgments.** We thank the anonymous reviewers for their helpful comments.

## References

1. Burkard, R.E., Dlaska, K., Klinz, B.: The quickest flow problem. *Mathematical Methods of Operations Research* **37**, 31–58 (1993)

2. Fleischer, L., Skutella, M.: Quickest flows over time. *SIAM Journal on Computing* **36**, 1600–1630 (2007)
3. Fleischer, L.K., Tardos, É.: Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters* **23**, 71–80 (1998)
4. Ford, L.R., Fulkerson, D.R.: *Flows in Networks*. Princeton University Press, Princeton (1962)
5. Groß, M., Kappmeier, J.-P.W., Schmidt, D.R., Schmidt, M.: Approximating Earliest Arrival Flows in Arbitrary Networks. In: Epstein, L., Ferragina, P. (eds.) *ESA 2012*. LNCS, vol. 7501, pp. 551–562. Springer, Heidelberg (2012)
6. Hausknecht, M., Au, T.-C., Stone, P., Fajardo, D., Waller, T.: Dynamic lane reversal in traffic management. In: *14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1929–1934 (2011)
7. Hirsch, M.D., Papadimitriou, C.H., Vavasis, S.A.: Exponential lower bounds for finding brouwer fix points. *Journal of Complexity* **5**, 379–416 (1989)
8. Hoppe, B., Tardos, É.: The quickest transshipment problem. *Mathematics of Operations Research* **25**, 36–62 (2000)
9. Hoppe, B.E.: *Efficient Dynamic Network Flow Algorithms*. PhD thesis, Cornell University (1995)
10. Kim, S., Shekhar, S.: Contraflow network reconfiguration for evaluation planning: A summary of results. In: *Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems*, pp. 250–259 (2005)
11. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences* **48**, 498–532 (1994)
12. Rebennack, S., Arulselvan, A., Elefteriadou, L., Pardalos, P.M.: Complexity analysis for maximum flow problems with arc reversals. *Journal of Combinatorial Optimization* **19**, 200–216 (2010)
13. Robbins, H.E.: A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly* **46**, 281–283 (1939)
14. Skutella, M.: An introduction to network flows over time. In: Cook, W., Lovász, L., Vygen, J. (eds.) *Research Trends in Combinatorial Optimization*, pp. 451–482. Springer (2009)
15. Tjandra, S.A.: *Dynamic network optimization with application to the evacuation problem*. PhD thesis, Technical University of Kaiserslautern (2003)
16. Tuydes, H., Ziliaskopoulos, A.: Network re-design to optimize evacuation contraflow. In: *Proceedings of the 83rd Annual Meeting of the Transportation Research Board*, Washington, DC (2004)
17. Tuydes, H., Ziliaskopoulos, A.: Tabu-based heuristic approach for optimization of network evacuation contraflow. *Transportation Research Record* 1964, 157–168 (2006)
18. Wolshon, B.: One-way-out: Contraflow freeway operation for hurricane evacuation. *Natural Hazards Review* **2**, 105–112 (2001)
19. Wolshon, B., Urbina, E., Levitan, M.: National review of hurricane evacuation plans and policies. Technical report, LSU Hurricane Center, Louisiana State University, Baton Rouge, Louisiana (2002)

# A Simple Efficient Interior Point Method for Min-Cost Flow

Ruben Becker<sup>1,2,3(✉)</sup> and Andreas Karrenbauer<sup>1,2</sup>

<sup>1</sup> Max Planck Institute for Informatics, Saarbrücken, Germany

<sup>2</sup> Max-Planck-Center for Visual Computing and Communication, Saarbrücken, Germany

<sup>3</sup> Saarbrücken Graduate School for Computer Science, Saarbrücken, Germany  
{ruben,karrenba}@mpi-inf.mpg.de

**Abstract.** We present a novel simpler method for the min-cost flow problem and prove that its expected running time is bounded by  $\tilde{O}(m^{3/2})$ . This matches the best known bounds, which have previously been achieved only by far more complex algorithms or by algorithms for special cases. Our contribution contains three algorithmic parts that are interesting in their own right: (1) We provide a linear time construction of an equivalent auxiliary network and interior primal and dual points, i.e. flows, node potentials and slacks, with potential  $P_0 = \tilde{O}(\sqrt{m})$ . (2) We present a potential reduction algorithm that transforms initial solutions of potential  $P_0$  to ones with duality gap below 1 in  $\tilde{O}(P_0 \cdot \text{CEF}(n, m, \varepsilon))$  time, where  $\varepsilon^{-1} = O(m^2)$  and  $\text{CEF}(n, m, \varepsilon)$  denotes the running time of any algorithm that computes an  $\varepsilon$ -approximate electrical flow. (3) We show that, taking solutions with duality gap less than 1 as input, one can compute optimal integral node potentials in  $O(m + n \log n)$  time with our novel crossover procedure. Altogether, using a variant of a state-of-the-art  $\varepsilon$ -electrical flow solver, we obtain a new simple algorithm for the min-cost flow problem running in  $\tilde{O}(m^{3/2})$ .

## 1 Introduction

The min-cost flow problem is one of the best studied problems in combinatorial optimization. Moreover, it represents an important special case of Linear Programming (LP) due to the integrality of the primal and dual polyhedra for arbitrary given integer costs  $c$ , capacities  $u$ , and demands  $b$ . That is, there are always integral primal and dual optimal solutions, provided that the problem is feasible and finite. It was first shown by Edmonds and Karp [1] in 1970 that these solutions can be computed in polynomial-time. Since then, there were many contributions to this problem. Due to space limitations, we can only mention some of the most important results such as the strongly polynomial time algorithm by Orlin [2] running in  $O(n^2 \log^2 n + nm \log n)$ .<sup>1</sup> Further scaling techniques like (generalized) cost-scaling were presented by Goldberg and Tarjan [3],

<sup>1</sup> We denote  $U := \|u\|_\infty$ ,  $C := \|c\|_\infty$  and  $\gamma := \max\{C, U\}$ . Moreover, we will use the  $\tilde{O}$ -notation to hide log-factors in  $n$ ,  $C$ ,  $U$  and  $\|b\|_\infty$ .

and the double scaling technique by Ahuja et al. [4], which yields a running time bound of  $O(nm \log \log U \log(nC))$ . However, all these algorithms failed to break through the barrier of  $n^2$ , even for sparse graphs, where  $m = O(n)$ . On the other hand, the asymptotically fastest known algorithms for solving general LPs are interior point methods. Karmarkar [5] presented the first poly-time interior point method running in  $O(n^{3.5}L^2)$  time, where  $n$  is the number of variables and  $L$  the size of the input. Later on, the work on interior point methods, and in particular, on the so-called potential reduction methods, was significantly advanced by Ye [6]. He presented an  $O(n^3L)$ -time algorithm. Vaidya [7] was the first to specialize interior point methods to network flow problems in 1989. He obtained a running time of  $O(n^2\sqrt{m} \log(n\gamma))$ , which matched the then best known bounds up to log-factors. Later on, there were many other contributions presenting specialized interior point methods for network flow problems. For the max-flow problem, Madry gave an elaborate interior point method [8] in 2008 that improved over a more than 30 year old bound. For the min-cost flow problem, Daitch and Spielman [9] gave a method running in  $\tilde{O}(m^{3/2})$  expected time. It is a dual central path following method and, in fact, solves a more general problem (lossy generalized flow) and thus is very involved and more technical than necessary for the classical min-cost flow problem. They use an efficient randomized solver for symmetric diagonally dominant (SDD) systems of linear equations based on the seminal work of Spielman and Teng [10] and later by Koutis et al. [11]. Recently, Kelner et al. [12] presented a simple, combinatorial, nearly-linear time algorithm for computing *approximate electrical flows* and thus also for approximately solving SDD systems. Motivated by their result, we show in this paper that such *approximate* electrical flows suffice for a potential reduction algorithm to obtain a much simpler min-cost flow algorithm running in  $\tilde{O}(m^{3/2})$  time, which matches the best known bounds up to log-factors. To this end, we also provide a new initialization technique and a novel crossover procedure, that both run in (near-)linear time and heavily exploit the network structure of the problem. We present a combinatorial view on potential reduction interior point methods for min-cost flow:

We, first of all, construct an auxiliary network with the same optimum, a primal interior solution (flows) and dual interior solutions (node potentials and dual slacks, i.e. reduced costs) for it. We then update these solutions iteratively by using electrical flow computations. To this end, we maintain a spanning tree of low stretch w.r.t. the resistances, which are determined by the current primal interior point, such that the smaller the value of the primal variable is, the higher the resistance of an arc gets. Now, an approximation of the electrical flow and corresponding node voltages are computed in this electrical network by augmenting electrical current along the fundamental cycles of the low-stretch tree in logarithmic time per augmentation. Unlike with the network simplex, the tree is not changed in every iteration, but only when the currents approximate the electrical flow sufficiently well, either a cycle update or a cut update of the interior point is performed, which then could lead to another tree that differs by more than one arc. In a cycle update, the flow is changed according to the

electrical currents. In a cut update, the slacks and node potentials are modified along cuts using the previously computed node voltages. The updates are such that a potential function decreases by at least some constant per step. This potential function is designed to serve in two ways: (1) when it drops below 0, the duality gap is smaller than 1 and we may stop since we are close to the optimal vertex, (2) it keeps us away from the boundary. We hereby take a shortcut through the polyhedron: instead of walking on the boundary as with most of the other combinatorial methods, e.g. minimum-mean cycle canceling, we warp to the basic solution corresponding to the new low-stretch tree. At the end, we apply our novel crossover procedure. It jumps from interior points of duality gap less than 1 to a solution with optimal integral node potentials in near-linear time. All running times in this paper are stated in terms of basic operations that also include comparisons in addition to the arithmetic operations. Due to space limitations, we have to omit some proofs in this version of the paper. They can be found in the full version.

### 1.1 Contribution

Our main contribution is a very simple algorithm for the min-cost flow problem terminating in  $O(m^{3/2}(\log \gamma + \log n) \log^3 n \log \log n) = \tilde{O}(m^{3/2})$  time, with high probability. In order to obtain this, our contribution contains the following three algorithmic parts that are already interesting in their own right.

1. We show that it suffices to compute primal and dual points with duality gap below 1, since our novel crossover procedure then finds optimal potentials in  $O(m + n \log n)$ .
2. We give a potential reduction method that, taking interior points of potential  $P_0$ , outputs interior points with duality gap below 1 in  $\tilde{O}(P_0 \cdot \text{CEF}(n, m, \varepsilon))$ , where  $\text{CEF}(n, m, \varepsilon)$  is the complexity of a certifying  $\varepsilon$ -electrical flow computation.
3. We give a method that, taking any min-cost flow problem as input, yields an auxiliary network with the same optimum and interior primal and dual points of potential  $P_0 = O(\sqrt{m}(\log \gamma + \log n))$  in linear time.

Our crossover procedure takes solutions with duality gap less than one and efficiently rounds the potentials to integral values. Using one max-flow computation in the admissible network, one can also obtain primal optimal solutions. This method does not rely on perturbation to remove degeneracy. However, if one prefers random perturbation over solving a max-flow problem (which does not increase the asymptotic running time), then the Separation Lemma [9, Lemma 3.13] can be applied at the beginning and our  $O(m + n \log n)$  time crossover will also identify the basic variables on its own additionally to the optimum dual solution, which is found even in the degenerate case. For the potential reduction method, we show how to use *approximate* electrical flow computations to reduce the duality gap of given primal and dual interior points of potential  $P_0$  below any constant  $c \in \mathbb{R}_{>0}$  in time  $\tilde{O}(P_0 \cdot \text{CEF}(n, m, \varepsilon))$ . We show that it suffices to

pick an  $\varepsilon$  such that  $\varepsilon^{-1}$  is polynomially bounded in  $n$  (i.e.  $\varepsilon^{-1} = O(m^2)$ ). Note that CEF typically scales logarithmically with  $\varepsilon^{-1}$  and thus its contribution is in  $O(\log n)$ . We also remove the necessity to compute the 2-norm of the primal update that is used for normalization in similar algorithms, as for example the one by Ye [6]. We show how to use the maximum of 1 and the  $\infty$ -norm instead of the 2-norm. In this way, our algorithm gets along with the field operations on rationals, similar to Kelner et al.'s.

### 1.2 The Min-Cost Flow Problem and Its Dual

In its most general form, the min-cost flow problem is stated as follows. Given a directed graph  $G = (V, A)$  with  $|V| = n$  and  $|A| = m$ , node demands  $b \in \mathbb{Z}^n$  with  $\mathbf{1}^T b = 0$ , arc costs  $c \in \mathbb{Z}^m$  and arc capacities  $u \in (\mathbb{N} \cup \{\infty\})^m$ , find a feasible flow  $x^* \in \mathbb{R}^m$ , i.e.  $0 \leq x^* \leq u$  and  $x^*(\delta^{\text{in}}(v)) - x^*(\delta^{\text{out}}(v)) = b_v$  for every  $v \in V$ ,<sup>2</sup> such that  $c^T x^* \leq c^T x$  for all feasible flows  $x$  or assert that no such flow exists. However, it is well-known, see e.g. [13], that this problem can be reduced to a setting without capacity constraints and only non-negative costs. Furthermore, we assume w.l.o.g. that the problem is feasible as well as finite from now on. We will discuss how to reduce the general problem to the setting used here in Section 4. For the time being, we write the problem as primal-dual pair

$$\min\{c^T x : Ax = b \text{ and } x \geq 0\} = \max\{b^T y : A^T y + s = c \text{ and } s \geq 0\},$$

where  $A \in \{-1, 0, 1\}^{n \times m}$  is the node-arc incidence matrix of  $G$ , i.e.  $A$  contains a column  $\alpha$  for every arc  $(v, w)$  with  $\alpha_v = -1$ ,  $\alpha_w = 1$  and  $\alpha_i = 0$  for all  $i \notin \{v, w\}$ .

## 2 Snapping to the Optimum

In this section, we show that solving the min-cost flow problem approximately, by the means of computing primal and dual solutions  $x$  and  $y^0, s^0$  of duality gap less than 1, is sufficient, since optimal integral potentials might be found in linear time then. The main underlying idea of our new linear time rounding procedure is the following. We iteratively construct sets  $S^k$ , starting with  $S^1 := \{s\}$  for an arbitrary vertex  $s$ . During one iteration  $k$ , we proceed as follows. Let us first assume  $b(S^k) < 0$ . Then, there has to be an outgoing arc from  $S^k$ , otherwise the problem would be infeasible. We enlarge  $S^k$  by the vertex  $w^k$  such that  $a^k = (v^k, w^k)$  for  $v^k \in S^k$  has minimal slack among all outgoing arcs from  $S^k$  and we increase the potentials  $y_w$  of all  $w \in V \setminus S^k$  by this minimal slack. It follows that the dual constraint of the arc  $a^k$  is satisfied with slack 0 and all other non-negativity constraints remain fulfilled. The objective value  $b^T y$  will be increased by this potential shift, since  $b(V \setminus S^k) > 0$ . In the case  $b(S^k) \geq 0$ , we decrease the potentials in  $V \setminus S^k$ , analogously by the minimum slack of all ingoing arcs. However to achieve a near-linear running time, these potential changes need to be performed in a lazy way. Using Fibonacci heaps, we can even reduce the running time to  $O(m + n \log n)$ . We give the pseudo-code of this method in Alg. 1 and show its correctness in Thm. 1.

<sup>2</sup> We write  $f(S) := \sum_{a \in S} f_a$  for  $S \subseteq A$  for any vector  $f \in \mathbb{R}^m$ .



**Algorithm 1.** Crossover

**Input** : Connected graph  $G = (V, A)$ , solution  $x$  and  $y^0, s^0$  in  $G$  with  $x^T s^0 < 1$ .  
**Output:** Optimal node potentials  $y$  in  $G$ .

Let  $s \in V$  be arbitrary and let  $\Delta^0 := -y_s^0, y_s \leftarrow 0, S^1 := \{s\}$ .

**for**  $k = 1, \dots, n - 1$  **do**  
     **if**  $b(S^k) < 0$  **or**  $\delta^{\text{in}}(S^k) = \emptyset$  **then**  
         Let  $\Delta^k = \min\{c_a + y_v - y_w^0 : a = (v, w) \in \delta^{\text{out}}(S^k)\}$   
         and  $a^k = (v^k, w^k) \in \delta^{\text{out}}(S^k)$  s.t.  $\Delta^k = c_{a^k} + y_{v^k} - y_{w^k}^0$ .  
     **else**  
         Let  $\Delta^k = -\min\{c_a + y_w^0 - y_v : a = (w, v) \in \delta^{\text{in}}(S^k)\}$   
         and  $a^k = (w_k, v_k) \in \delta^{\text{in}}(S^k)$  s.t.  $\Delta^k = -c_{a^k} - y_{w^k}^0 + y_{v^k}$ .  
      $y_{w^k} \leftarrow y_{w^k}^0 + \Delta^k, S^{k+1} \leftarrow S^k \cup \{w^k\}$

**return** potentials  $y$ .

**Theorem 1.** *Let Alg. 1 be initialized with primal and dual solutions  $x$  and  $y^0, s^0$  with  $x^T s^0 < 1$ . Then Alg. 1 outputs optimal integral potentials  $y$  in  $O(m + n \log n)$ .*

*Proof.* We assume w.l.o.g. that the vertices are labeled  $1, \dots, n$  in the order in which they are added to  $S$ . We show, by induction, that the potentials

$$y_v^k = \begin{cases} y_v^0 + \Delta^{k-1}, & k \leq v \\ y_v^{k-1}, & k > v \end{cases} \quad \text{are feasible, i.e. } s_a^k := c_a + y_v^k - y_w^k \geq 0 \quad \forall a = (v, w).$$

For the induction base, we note that  $y_v^1$  is just  $y_v^0$  shifted by  $\Delta^0 = -y_s^0$  and hence it constitutes valid potentials. For the inductive step let us consider iteration  $k > 1$  and let  $a = (v, w)$  be an arbitrary arc. Let  $i := \min\{v, w\}$  and  $j := \max\{v, w\}$ . With the convention  $c_{(j,i)} = -c_{(i,j)}$  and thus  $s_{(j,i)}^k = -s_{(i,j)}^k$ , we obtain

$$s_{(i,j)}^k = c_{(i,j)} + y_i^k - y_j^k = c_{(i,j)} + \begin{cases} y_i^0 - y_j^0, & k \leq i \\ y_i^{k-1} - (y_j^0 + \Delta^{k-1}), & i < k \leq j \\ y_i^{k-1} - y_j^{k-1}, & j < k \end{cases}$$

For the first and third case, we apply the induction hypothesis and obtain  $s_a^k \geq 0$ . For the second case, we first note that

$$\begin{aligned} \Delta^{k-1} &= \sigma \cdot c_{a^{k-1}} + y_{v^{k-1}} - y_{w^{k-1}}^0 \text{ where } \sigma = \begin{cases} 1 & \text{if } b(S^{k-1}) < 0 \text{ or } \delta^{\text{in}}(S^{k-1}) = \emptyset \\ -1 & \text{otherwise} \end{cases} \\ &= \sigma \cdot c_{a^{k-1}} + (y_{v^{k-1}} - \Delta^{k-2}) - y_{w^{k-1}}^0 + \Delta^{k-2} = \sigma \cdot s_{a^{k-1}}^{k-1} + \Delta^{k-2} \end{aligned}$$

Since  $i < k \leq j$  and thus  $(i, j) \in \delta^{\text{out}}(S^{k-1})$ , this yields

$$s_{(i,j)}^k = c_{(i,j)} + y_i^{k-1} - (y_j^0 + \Delta^{k-2}) - \sigma \cdot s_{a^{k-1}}^{k-1} = s_{(i,j)}^{k-1} - \sigma \cdot s_{a^{k-1}}^{k-1}.$$

Independent of  $a$  being  $(i, j)$  or  $(j, i)$ , we get  $s_a^k = s_a^{k-1} \pm s_{a^{k-1}}^{k-1} \geq 0$  since  $a^{k-1}$  is a minimizer and by the non-negativity of the slacks due to the induction hypothesis. Hence, the output potentials are *feasible*. In addition, we construct one tight constraint in each iteration, since  $s_a^k = 0$  if  $a = a^{k-1}$ . Since  $y_s = 0$  and  $c \in \mathbb{Z}^m$ , we conclude that after termination  $y$  is *integral*. Note that the optimum objective value is integer and thus  $\lceil b^T y^0 \rceil$  because  $x^T s^0 < 1$ . We have

$$\begin{aligned} b^T y^k - b^T y^{k-1} &= \sum_{v \in V} b_v y_v^k - \sum_{v \in V} b_v y_v^{k-1} = \sum_{v \geq k} b_v (y_v^0 + \Delta^{k-1}) - \sum_{v \geq k} b_v y_v^{k-1} \\ &= \sum_{v \geq k} b_v y_v^0 + \sum_{v \geq k} b_v \Delta^{k-1} - \sum_{v \geq k} b_v y_v^0 - \sum_{v \geq k} b_v \Delta^{k-2} \\ &= -\sigma \cdot s_{a^{k-1}}^{k-1} \cdot b(S^{k-1}) \geq 0, \end{aligned}$$

because  $\delta^{\text{in}}(S^{k-1}) = \emptyset$  implies that  $b(S^{k-1}) \leq 0$  or that the instance is infeasible. Since  $b, y \in \mathbb{Z}^n$  and  $b^T y - \lceil b^T y^0 \rceil < 1$  we have that  $y$  is *optimal*. A similar implementation as used for Dijkstra’s algorithm but with two Fibonacci Heaps, one for the nodes adjacent to  $S^k$  through  $\delta^{\text{in}}(S^k)$  and  $\delta^{\text{out}}(S^k)$  each, yields the run time of  $O(m + n \log n)$ .  $\square$

### 3 Potential Reduction Algorithm

We will now describe our Potential Reduction Algorithm, which is inspired by Ye’s algorithm [6]. It maintains a primal solution  $x$  and dual slacks  $s$ . We evaluate such a pair by the *potential function*  $P(x, s) := q \ln(x^T s) - \sum_{a \in A} \ln(x_a s_a) - m \ln m$  for some scalar  $q = m + p \in \mathbb{Q}$  to be chosen later. Note that the duality gap  $x^T s = c^T x - b^T y$  serves as measure for the distance to optimality of  $x$  and  $s$ . An equivalent formulation of the potential function yields

$$P(x, s) = p \ln(x^T s) + m \ln \left( \frac{1}{m} \sum_{a \in A} x_a s_a \right) - m \ln \sqrt[m]{\prod_{a \in A} x_a s_a} \geq p \ln(x^T s), \quad (1)$$

because the arithmetic mean is bounded by the geometric mean from below. Thus,  $P(x, s) < 0$  implies  $x^T s < 1$ . As we have shown in Section 2, solutions satisfying  $x^T s < 1$  can be efficiently rounded to integral optimal solutions. Thus, we follow the strategy to minimize the potential function by a combinatorial gradient descent until the duality gap drops below 1. To this end, we shall project the gradient  $g := \nabla_x P = \frac{q}{x^T s} s - X^{-1} \mathbf{1}$ , where  $X := \text{diag}(x)$ , on the cycle space of the network. However, we do not use the standard scalar product for the projection but a skewed one as it is common in the literature on interior point methods. This skewed scalar product may also be considered as the standard one in a scaled space where  $x$  is mapped to  $X^{-1} x = \mathbf{1}$ . By setting  $s' := X s$ , the duality gap  $x^T s = \mathbf{1}^T s'$  and the potential function  $P(x, s) = P(\mathbf{1}, s')$  remain unchanged. Accordingly, we define  $\hat{A} := AX$  and  $g' := \nabla_x P|_{x=\mathbf{1}, s=s'} = X g$ .

We start with given initial primal and dual solutions  $x, s$  or rather with their analogs  $\mathbf{1}, s'$  in the scaled space, which may be found for example with our

initialization method described in Section 4. Now, it would be desirable to move the scaled flows  $x'$  in the direction of  $-g'$ , the direction of steepest descent of the potential function. However,  $g'$  may not be a feasible direction, i.e. might not be a cycle<sup>3</sup>, since  $\bar{A}g' \neq 0$  in general. Thus, we wish to find a direction  $d'$  in the kernel of  $\bar{A}$ , i.e. a cycle that is closest to  $g'$ . Computing  $d'$  amounts to solve the convex optimization problem

$$\min\{\|g' - d'\|_2^2 : \bar{A}d' = 0\} = \min\{\|f\|_R^2 : Af = \chi\}, \tag{2}$$

where we set  $f = X(g' - d')$ ,  $R = X^{-2}$  and  $\chi = \bar{A}g'$ . The latter is actually an electrical flow problem. We briefly review electrical flows, for more details, see for example [8, 12].

**Electrical Flows.** Let  $\chi \in \mathbb{Q}^n$  be a *current source* vector with  $\mathbf{1}^T\chi = 0$  and let  $r \in \mathbb{Q}_{\geq 0}^m$  be a resistance vector on the arcs, denote  $R = \text{diag}(r)$  and  $\|v\|_R := \sqrt{v^T R v}$  for  $v \in \mathbb{R}^m$ .

**Definition 1 (Electrical Flow).** Let  $\chi \in \mathbb{Q}^n$  with  $\mathbf{1}^T\chi = 0$ .

1. The unique flow  $f^* \in \mathbb{Q}^m$  with  $\|f^*\|_R^2 := \min\{\|f\|_R^2 : Af = \chi\}$  is the electrical flow.
2. Let  $\varepsilon \geq 0$  and  $f \in \mathbb{R}^m$  with  $Af = \chi$  and  $\|f\|_R^2 \leq (1 + \varepsilon)\|f^*\|_R^2$ , then  $f$  is called an  $\varepsilon$ -electrical flow.
3. Let  $s$  be a fixed node,  $T$  a spanning tree,  $P(s, v)$  the unique path in  $T$  from  $s$  to  $v$  and  $f \in \mathbb{R}^m$ . The tree induced voltages  $\pi \in \mathbb{R}^n$  are defined by  $\pi(v) := \sum_{a \in P(s, v)} f_a r_a$ .
4. For any  $a = (v, w) \in A \setminus T$ , we define  $C_a := \{a\} \cup P(w, v)$  and  $r(C_a) := \sum_{b \in C_a} r_b$ . We write  $\tau(T) := \sum_{a \in A \setminus T} r(C_a)/r_a$  for the tree condition number of  $T$ .

The dual of the electrical flow problem is  $\max\{2\pi^T\chi - \pi^T A R^{-1} A^T \pi : \pi \in \mathbb{R}^n\}$ , where  $\pi$  are called *voltages*. We conclude that an optimal solution  $\pi^*$  satisfies  $A R^{-1} A^T \pi^* = \chi$ .

**Definition 2 (Certifying  $\varepsilon$ -Electrical Flow Algorithm).** Let  $\varepsilon > 0$ . A certifying  $\varepsilon$ -electrical flow algorithm is an algorithm that computes an  $\varepsilon$ -electrical flow  $f$  and voltages  $\pi \in \mathbb{Q}^n$  such that  $\|\pi - \pi^*\|_{A R^{-1} A^T}^2 \leq \varepsilon \|\pi^*\|_{A R^{-1} A^T}^2$ , where  $\pi^*$  is an optimal dual solution. We define  $\text{CEF}(n, m, \varepsilon)$  to be a bound on the running time of a certifying  $\varepsilon$ -electrical flow algorithm for directed graphs with  $n$  nodes and  $m$  arcs.

Kelner et al. [12] present a simple  $\varepsilon$ -electrical flow algorithm with expected approximation guarantee. However, we transform their algorithm to one with an exact approximation guarantee and linear running time with high probability. Similarly to them, we compute a low-stretch spanning tree  $T$  (w.r.t. the

---

<sup>3</sup> The kernel of  $\bar{A}$ , up to the scaling with  $X$ , corresponds to the cycle space of the graph.

resistances), which has tree condition number  $\tau(T) = O(m \log n \log \log n)$  using the method of Abraham and Neiman [14] that runs in  $O(m \log n \log \log n)$ . We then sample non-tree edges  $a$  according to the same probability distribution  $p_a := \frac{1}{\tau(T)} \frac{r(C_a)}{r_a}$  and push flow along the cycle  $C_a$  until the gap between primal and dual objective value becomes less than  $\varepsilon$ . The running time of this approach is  $O(m \log^2 n \log(n/\varepsilon) \log \log n) = \tilde{O}(m)$  for  $\varepsilon^{-1} = O(\text{poly}(n))$  with high probability as we show in Thm. 2. Note that it suffices for our purpose to mimic their `SimpleSolver`, which scales with  $\log(n/\varepsilon)$  instead of  $\log(1/\varepsilon)$  as their improved version does. We remark that, as in their solver, the flow and voltage updates should be performed using a special tree data structure [12, Section 5], which allows updating the flow in  $O(\log n)$ . Moreover, gap should only be computed every  $m$  iterations, which results in  $O(1)$  amortized time per iteration for the update of gap.

**The Method.** Using any certifying  $\varepsilon$ -electrical flow algorithm, we can compute an approximation of  $d'$  by solving problem (2) and obtain an  $\varepsilon$ -electrical flow  $f$ . We will now describe how to use this *approximate* electrical flow for our updates. In our electrical flow problem the resistances  $R$  are given by  $X^{-2}$  and the current sources  $\chi$  by  $\bar{A}g'$ . We compute a cycle  $\hat{x}' = g' - X^{-1}f$  from the flow as well as a cut  $\hat{s}' = \bar{A}^T\pi$  from the voltages  $\pi$ . Note that  $\hat{x}_a = g_a/r_a - f_a$  and  $\hat{s}_a = \pi_w - \pi_v$ , so  $\hat{x}_a$  is computed using the battery divided by the resistance of an arc subtracted by the electrical flow, whereas  $\hat{s}_a$  is just the voltage difference of the two nodes. The idea is to push flow around the cycle  $\hat{x}'$  in a primal step, whereas, in a dual step, we modify the slacks and node potentials corresponding to the cut  $\hat{s}'$ . In usual gradient descent methods, like for example in Ye's algorithm the decision whether to make a primal or dual step is made dependent on  $\|d'\|_2$ , the length of the projection of the gradient on the null-space. In our setting, however, we do not know the exact projection  $d'$  of  $g'$ . Nevertheless, we can show that the 2-norm of  $z' = g' - \hat{s}'$  does not differ too much from  $\|d'\|_2$ , so deciding dependent on  $\|z'\|_2^2$  is possible. We note that another crucial difference between our Potential Reduction Algorithm and other algorithms, as for example Ye's algorithm, is that we normalize the cycle  $\hat{x}'$  by  $\max\{1, \|\hat{x}'\|_\infty\}$ , whereas in Ye's algorithm the normalization is done with  $\|\hat{x}'\|_2$ .

We remark that our method works with any certifying  $\varepsilon$ -electrical flow algorithm. However, we merge the version of the `SimpleSolver` of Kelner et al. [12] as described above in our pseudocode implementation of Alg. 2 to be more self-contained.

**Analysis.** It is not hard to see that that the primal and dual steps in the algorithm are in fact feasible moves, for a proof see the full version of the paper.

**Lemma 1.** *The new iterates  $\bar{x} = X(\mathbb{1} - \lambda \frac{\hat{x}'}{\max\{1, \|\hat{x}'\|_\infty\}})$ ,  $\bar{y} = y + \mu\pi$  and  $\bar{s} = X^{-1}(s' - \mu\hat{s}')$  are primal and dual feasible solutions.*

---

**Algorithm 2.** Potential Reduction Algorithm

---

**Input** : Feasible flow  $x > 0$ , feasible dual variables  $y$  and  $s > 0$ , parameter  $\delta$

**Output**: Feasible flow  $x > 0$ , feasible dual variables  $y$  and  $s > 0$  s.t.  $x^T s < 1$ .

**while**  $x^T s \geq 1$  **do**

$g' := \frac{q}{x^T s} X s - \mathbf{1}$ ,  $\chi := \bar{A} g'$ ,  $r := X^{-2} \mathbf{1}$

/\*  $\varepsilon$ -electrical flow computation, similar to SimpleSolver in [12] \*/

$T :=$  low-stretch spanning tree w.r.t.  $r$ ,  $\tau(T) := \sum_{a \in A \setminus T} \frac{r(C_a)}{r_a}$ ,  $p_a := \frac{r(C_a)}{\tau(T)r_a}$

$f :=$  tree solution with  $Af = \chi$  for  $T$ ,  $\pi :=$  tree induced voltages of  $f$

$\text{gap} := f^T R f - 2\pi^T \chi + \pi^T \bar{A} \bar{A}^T \pi$

**repeat**

Randomly sample  $a \in A \setminus T$  with probability  $p_a$

Update  $f$  by pushing  $\sum_{b \in C_a} r_b f_b / r(C_a)$  flow through  $C_a$  in the direction of  $a$

Update voltages  $\pi$  and occasionally compute gap

**until**  $\text{gap} < \delta$

/\* Make a primal or dual update. \*/

Set  $\hat{x}' := g' - X^{-1} f$ ,  $\hat{s}' = \bar{A}^T \pi$  and  $z' = g' - \hat{s}'$ .

**if**  $\|z'\|_2^2 \geq 1/4$  **then**

Cycle update:  $x' := \mathbf{1} - \lambda \frac{\hat{x}'}{\max\{1, \|\hat{x}'\|_\infty\}}$ , where  $\lambda = 1/4$ .

**else**

Cut update:  $s' := s' - \mu \hat{s}'$  and  $y := y + \mu \pi$ , where  $\mu = \frac{\mathbf{1}^T s'}{q}$ .

**return**  $x$  and  $y, s$

---

The following lemma, whose proof may also be found in the full version, shows that the potential is reduced by a constant amount in each step. We remark that although the proof for the dual step is essentially similar to the proof for Ye’s algorithm, the normalization with the  $\infty$ -norm in the primal step and the fact that  $\hat{x}'$  is only an approximation of  $d'$  requires non-trivial changes in the proof for the primal step.

**Lemma 2.** *If  $\delta \leq 1/8$  and  $p^2 \geq m \geq 4$ , the potential reduction is constant in each step, more precisely it holds that*

$$P(\mathbf{1}, s') - P\left(\mathbf{1} - \lambda \frac{\hat{x}'}{\max\{1, \|\hat{x}'\|_\infty\}}, s'\right) \geq 1/64 \quad \text{and} \quad P(\mathbf{1}, s') - P(\mathbf{1}, s' - \mu \hat{s}') \geq \frac{1}{12}.$$

We already remarked that  $P(x, s) < 0$  implies  $x^T s < 1$ , hence  $P(x, s) \geq 0$  holds throughout the algorithm. With Lemma 2, the initial potential bounds the number of iterations.

**Theorem 2.** *Given primal and dual interior points with potential  $P_0$  as input, Alg. 2 outputs interior primal and dual solutions  $x$  and  $y, s$  with  $x^T s < 1$  after  $O(P_0)$  iterations. It can be implemented such that it terminates after  $O(P_0 \cdot m \log^3(m) \log \log m)$  time with probability at least  $1 - \exp(-m \log^3(m) \log \log m)$ .*

The proof of this theorem may be found in the full version of the paper. The theorem follows by an application of Markov’s inequality to the probability that the gap is larger than  $\delta$  after one iteration of the Repeat-Until loop. We remark that we can also keep running the algorithm until  $x^T s < c$  for any  $c \in \mathbb{R}_{\geq 0}$  without affecting the running time. In addition, we get the following more general result. To prove it, it remains to show that a  $1/(16q^2)$ -electrical flow fulfills  $\text{gap} \leq 1/8$ .

**Theorem 3.** *Given primal and dual interior points with a potential of  $P_0$  as input, there is an algorithm that outputs interior primal and dual solutions  $x$  and  $y, s$  with  $x^T s < 1$  and needs  $O(P_0 \cdot \text{CEF}(n, m, 1/(16q^2)))$  time, where  $q = m + \min\{k \in \mathbb{Z} : k^2 \geq m\}$  and  $\text{CEF}(n, m, \varepsilon)$  is the running time of any certifying  $\varepsilon$ -electrical flow algorithm.*

*Proof.* It remains to show that a  $1/(16q^2)$ -electrical flow fulfills  $\text{gap} \leq 1/8$ . The approximation guarantee from the certifying  $\varepsilon$ -electrical flow algorithm for the primal and dual solution yield

$$\|g' - \hat{x}'\|_2^2 \leq (1 + \varepsilon)\|g' - d'\|_2^2 \quad \text{and} \quad \|\pi - \pi^*\|_{\bar{A}\bar{A}^T}^2 \leq \varepsilon\|\pi^*\|_{\bar{A}\bar{A}^T}^2, \tag{3}$$

the second guarantee equivalently writes as

$$\varepsilon\|g' - d'\|_2^2 \geq \|z' - d'\|_2^2 = \|z'\|_2^2 - 2d'^T(g' - \bar{A}^T \pi) + \|d'\|_2^2 = \|z'\|_2^2 - \|d'\|_2^2.$$

Together with (3), we obtain

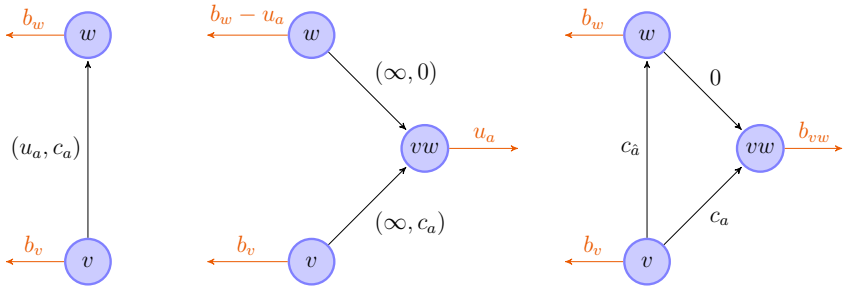
$$\begin{aligned} \text{gap} &= f^T Rf - 2\pi^T \chi + \pi^T \bar{A}\bar{A}^T \pi = \|g' - \hat{x}'\|_2^2 - 2g'^T(g' - z') + \|g' - z'\|_2^2 \\ &= \|g' - \hat{x}'\|_2^2 - \|g'\|_2^2 + \|z'\|_2^2 \leq (1 + \varepsilon)\|g' - d'\|_2^2 - \|g'\|_2^2 + \varepsilon\|g' - d'\|_2^2 + \|d'\|_2^2 \\ &\leq 2\varepsilon\|g' - d'\|_2^2 \leq 2\varepsilon\|g'\|_2^2 \leq 2\varepsilon q^2 \leq \frac{1}{8}. \end{aligned}$$

### 4 Initialization

In this section, we describe how to find initial points with  $P_0 = \tilde{O}(\sqrt{m})$  that we can use to initialize Alg. 2. We assume w.l.o.g. that the given min-cost flow instance is finite, that the capacities are finite and that the costs are non-negative. In order to be self-contained, we also justify these assumptions in the full version of the paper. We remark that we do not need to check feasibility, since the crossover procedure presented above enables us to recognize infeasibility. This is described at the end of this section.

**Removing Capacity Constraints.** Using a standard reduction, we modify the network in order to get rid of the upper bound constraints  $x \leq u$ . Let  $G_0 = (V_0, A_0)$  denote the original input graph. For an edge  $a = (v, w) \in A_0$ , we proceed as follows, see from left to middle in Figure 1: Remove  $a$ , insert a node  $vw$ , insert arcs  $\acute{a} = (v, vw)$  and  $\grave{a} = (w, vw)$  with  $c_{\acute{a}} = c_a$  and  $c_{\grave{a}} = 0$ , respectively.<sup>4</sup> Moreover, set  $b_{vw} = u_a$  and subtract  $u_a$  from  $b_w$ .

<sup>4</sup> We remark that the accents reflect the direction in which the arc is drawn in Figure 1.



**Fig. 1.** The transition from the left to the middle, which is done for each arc, removes the capacity constraint. From the middle to the right: In order to balance the  $x_a s_a$ , we introduce the arc  $\hat{a} = (v, w)$  with high cost  $c_{\hat{a}}$  and reroute flow along it. The direction of  $\hat{a}$  depends on a tree solution  $z$  in  $G_0$ . It is flipped if  $z_a \leq u_a/2$ .

**Finding the Initial Flow.** Recall the equivalent form of the potential function in (1). It illustrates that the potential becomes small if the ratio between the arithmetic and the geometric mean does, this in turn is the case if the variance of the  $x_a s_a$  is low over all  $a \in A$ . This observation is crucial for our routine Alg. 3 that finds an initial flow with low potential. Our aim is to balance the flows on  $(v, vw)$  and  $(w, vw)$ , by introducing the arc  $(v, w)$  or  $(w, v)$ , see Figure 1 from middle to right. Since we perform the two transitions of Figure 1 together, we will refer to the resulting graph as  $G_1 = (V_1, A_1)$  with  $|V_1| = n_1$  and  $|A_1| = m_1 = 3m$ . For the sake of presentation, we assume w.l.o.g. that all capacities  $u_a$  of the original graph  $G$  were odd such that  $z_a - u_a/2 \neq 0$  for all integers  $z_a$ .<sup>5</sup>

---

**Algorithm 3.** Balance Arcs

---

**Input** :  $G_0 = (V_0, A_0)$ , parameter  $t$ .  
**Output**: Graph  $G_1$ , primal and dual solutions  $x$ , and  $y, s$ , such that  $x_a s_a \in [t, t + CU/2]$ .  
 Compute a tree solution in  $G_0$ , obtain integral (not necessarily feasible) flow  $z$ .  
**for every arc**  $a \in A_0$  **do**  
     Insert node  $vw$ , arcs  $\acute{a} = (v, vw)$ ,  $\grave{a} = (w, vw)$  with  $c_{\acute{a}} = c_a$ ,  $c_{\grave{a}} = 0$ , set  $x_{\acute{a}} = x_{\grave{a}} = u_a/2$   
     **if**  $z_a > u_a/2$  **then**  
         Replace  $a$  by  $\acute{a} = (v, w)$   
     **else**  
         Replace  $a$  by  $\grave{a} = (w, v)$   
      $c_{\hat{a}} = \lceil t/|z_a - u_a/2| \rceil$ ,  $x_{\hat{a}} := |z_a - u_a/2|$ ,  $y_{vw} := -2t/u_a$  and  $y_v, y_w := 0$   
**return** the resulting graph  $G_1 = (V_1, A_1)$  and  $x, y$  with corresponding slacks  $s$ .

---

<sup>5</sup> This is justified by the following argument: If the capacity of an arc is even, then we add a parallel arc of capacity 1 and reduce the capacity of the original arc by 1.

**Theorem 4.** Let  $G_1 = (V_1, A_1)$ ,  $x, y, s$  be output by Alg. 3, let  $\Gamma := \max\{C, U, \|b\|_1/2\}$ . Then, it holds that  $x_a s_a \in [t, t + \Gamma^2]$  for all  $a \in A_1$ . Furthermore, setting  $t = m\Gamma^3$  and  $p = \min\{k \in \mathbb{Z} : k^2 \geq m_1\}$  yields  $P(x, s) = O(\sqrt{m} \log(n\gamma))$ .

*Proof.* Let  $a \in A_0$  be any arc in  $G_0$ . We have  $x_{\hat{a}} = x_a = u_a/2$ . It holds that,

$$x_{\hat{a}} s_{\hat{a}} = \frac{u_a}{2} \left( c_{\hat{a}} + \frac{2t}{u_a} \right) = t + \frac{u_a c_a}{2} \leq t + \Gamma^2, \quad x_{\hat{a}} s_{\hat{a}} = \frac{u_a}{2} \left( c_{\hat{a}} + \frac{2t}{u_a} \right) = t \quad \text{and}$$

$$x_{\hat{a}} s_{\hat{a}} \geq \left| z_a - \frac{u_a}{2} \right| \frac{t}{\left| z_a - \frac{u_a}{2} \right|} = t \quad \text{and} \quad x_{\hat{a}} s_{\hat{a}} \leq \left| z_a - \frac{u_a}{2} \right| \left( \frac{t}{\left| z_a - \frac{u_a}{2} \right|} + 1 \right) \leq t + \Gamma.$$

We now consider the potential function with  $q = m_1 + p$ , we will fix  $p$  below:

$$P(x, s) := q \ln(x^T s) - \sum_{a \in A_1} \ln(x_a s_a) - m_1 \ln m_1 \leq q \ln(m_1 t + 2m\Gamma^2) - m_1 \ln m_1 t$$

$$\leq q \ln \left( 1 + \frac{m_1 \Gamma^2}{m_1 t} \right) + p \ln m_1 t \leq \frac{m_1 + p}{m\Gamma} + p \ln(m_1^2 \Gamma^3), \text{ since } t = m\Gamma^3.$$

For  $p = \min\{z \in \mathbb{Z} : z^2 \geq m_1\}$ , we get  $P(x, s) = O(\sqrt{m} \log n\Gamma) = O(\sqrt{m} \log n\gamma)$ . □

Alg. 3 can be implemented in  $O(m)$  time. We remark that, due to the high costs of the arcs  $A_1 \setminus A_0$ , there will never be flow on them in an optimal solution. In particular, these arcs are more expensive than any path in the original network because  $c_{\hat{a}} = \lceil t/|z_a - u_a/2| \rceil \geq mCU$ . Therefore, the optimum of the problem is not changed by the introduction of the arcs  $\hat{a}$ . We remark that the resulting network is always feasible. This is why we can assume feasibility in Section 2.

**Summary.** We first run Alg. 3 on the input graph  $G_0$  to construct the auxiliary network  $G_1$ . We then initialize Alg. 2 with the obtained interior points. If  $\lceil b^T y^0 \rceil > mCU$ , the problem in  $G_0$  was infeasible, since any solution in  $G_0$  is bounded by  $mCU$ . Otherwise, we apply Alg. 1 and obtain optimal integral optimal potentials  $y$  in  $G_1$ . Let  $H_1$  be the admissible network, i.e. the graph  $G_1$  with all arcs with slack 0. Consider  $H_0$ , the graph resulting by removing all arcs  $\hat{a}$  from  $H_1$  that were introduced by Alg. 3. By a max-flow computation we compute a feasible solution  $x$  in  $H_0$ , which is optimal in  $G_0$  by complementary slackness  $x$ . If  $H_0$  is however infeasible, there is a set  $S$  with  $b(S) \leq -1$  and  $\delta_{H_0}^{\text{out}}(S) = \emptyset$  [15, Corollary 11.2h]. Since  $y$  is optimal in  $G_1$ , there is an arc  $\hat{a} \in \delta_{G_1}^{\text{out}}(S)$  with  $s_{\hat{a}} = 0$ , thus  $\hat{a} \in \delta_{H_1}^{\text{out}}(S)$ . It follows that there is always a feasible and integral solution  $z$  in  $H_1$  with  $z_{\hat{a}} \geq 1$  that is optimal in  $G_1$ . With  $c_{\hat{a}} \geq mCU$ , we conclude that the cost of  $z$  is larger than  $mCU$ , which contradicts  $\lceil b^T y^0 \rceil \leq mCU$ . Since the max-flow computation requires  $O(m^{3/2} \log(n^2/m) \log U)$  if it is carried out with the algorithm of Goldberg and Rao [16], this yields the overall run-time.

## References

1. Edmonds, J., Karp, R.M.: Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. In: Combinatorial Structures and Their Applications, pp. 93–96. Gordon and Breach, New York (1970)



2. Orlin, J.B.: A Faster Strongly Polynomial Minimum Cost Flow Algorithm. In: Simon, J. (ed.) STOC, pp. 377–387. ACM (1988)
3. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.* **15**, 430–466 (1990)
4. Ahuja, R.K., Goldberg, A.V., Orlin, J.B., Tarjan, R.E.: Finding minimum-cost flows by double scaling. *Math. Program.* **53**, 243–266 (1992)
5. Karmarkar, N.: A New Polynomial-Time Algorithm for Linear Programming. In: DeMillo, R.A. (ed.) STOC 1984, pp. 302–311. ACM (1984)
6. Ye, Y.: An  $O(n^3L)$  potential reduction algorithm for linear programming. *Math. Program.* **50**, 239–258 (1991)
7. Vaidya, P.M.: Speeding-Up Linear Programming Using Fast Matrix Multiplication (Extended Abstract). In: FOCS, pp. 332–337. IEEE Computer Society (1989)
8. Mađry, A.: Navigating Central Path with Electrical Flows: from Flows to Matchings, and Back. In: 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS) (2013)
9. Daitch, S.I., Spielman, D.A.: Faster approximate lossy generalized flow via interior point algorithms. In: Dwork, C. (ed.) STOC, pp. 451–460. ACM (2008)
10. Spielman, D.A., Teng, S.H.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In Babai, L. (ed.) STOC, pp. 81–90. ACM (2004)
11. Koutis, I., Miller, G.L., Peng, R.: Approaching Optimality for Solving SDD Linear Systems. In: FOCS, pp. 235–244. IEEE Computer Society (2010)
12. Kelner, J.A., Orecchia, L., Sidford, A., Zhu, Z.A.: A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) STOC, pp. 911–920. ACM (2013)
13. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network flows - Theory. Algorithms and Applications.* Prentice Hall (1993)
14. Abraham, I., Neiman, O.: Using Petal-Decompositions to Build a Low Stretch Spanning Tree. In: Karloff, H.J., Pitassi, T. (eds.) STOC, pp. 395–406. ACM (2012)
15. Schrijver, A.: *Combinatorial Optimization: Polyhedra and Efficiency. Algorithms and combinatorics.* Springer (2003)
16. Goldberg, A.V., Rao, S.: Beyond the Flow Decomposition Barrier. In: FOCS, pp. 2–11 (1997)

# Decremental All-Pairs ALL Shortest Paths and Betweenness Centrality

Meghana Nasre<sup>1</sup>(✉), Matteo Pontecorvi<sup>2</sup>(✉), and Vijaya Ramachandran<sup>2</sup>

<sup>1</sup> Indian Institute of Technology, Madras, India  
meghana@cse.iitm.ac.in

<sup>2</sup> University of Texas at Austin, Austin, USA  
{cavia,vlr}@cs.utexas.edu

**Abstract.** We consider the all pairs all shortest paths (APASP) problem, which maintains the shortest path dag rooted at every vertex in a directed graph  $G = (V, E)$  with positive edge weights. For this problem we present a decremental algorithm (that supports the deletion of a vertex, or weight increases on edges incident to a vertex). Our algorithm runs in amortized  $O(\nu^{*2} \cdot \log n)$  time per update, where  $n = |V|$ , and  $\nu^*$  bounds the number of edges that lie on shortest paths through any given vertex. Our APASP algorithm can be used for the decremental computation of betweenness centrality (BC), which is widely used in the analysis of large complex networks. No nontrivial decremental algorithm for either problem was known prior to our work. Our method is a generalization of the decremental algorithm of Demetrescu and Italiano [3] for unique shortest paths, and for graphs with  $\nu^* = O(n)$ , we match the bound in [3]. Thus for graphs with a constant number of shortest paths between any pair of vertices, our algorithm maintains APASP and BC scores in amortized time  $O(n^2 \cdot \log n)$  under decremental updates, regardless of the number of edges in the graph.

## 1 Introduction

Given a directed graph  $G = (V, E)$ , with a positive real weight  $\mathbf{w}(e)$  on each edge  $e$ , we consider the problem of maintaining the shortest path dag rooted at every vertex in  $V$  (we will refer to these as the *SP dags*). We use the term *all-pairs ALL shortest paths (APASP)* to denote the collection of SP dags rooted at all  $v \in V$ , since one can generate all the (up to exponential number of) shortest paths in  $G$  from these dags. These dags give a natural structural property of  $G$  which is of use in any application where several or all shortest paths need to be examined. A particular application that motivated our work is the computation of betweenness centrality (BC) scores of vertices in a graph [4].

In this paper we present a decremental algorithm for the APASP problem, where each update in  $G$  either deletes or increases the weight of some edges

---

This work was supported in part by NSF grant CCF-0830737. The first author was also supported by CSE/14-15/824/NFIG/MEGA. The second and third authors were also supported by NSF grant CCF-1320675.

incident on a vertex. Our method is a generalization of the method developed by Demetrescu and Italiano [3] (the ‘DI’ method) for decremental APSP where only one shortest path is needed. The DI algorithm [3] runs in  $O(n^2 \cdot \log n)$  amortized time per update, for a sufficiently long update sequence. In [3] the result is extended to a fully dynamic algorithm that runs in  $O(n^2 \cdot \log^3 n)$  time, and this result was improved to  $O(n^2 \cdot \log^2 n)$  amortized time by Thorup [14]. We briefly discuss the fully dynamic case at the end of our paper. In these earlier algorithms, the unique shortest paths assumption is crucial.

In addition to APASP, our method gives decremental algorithms for the following two problems.

**Locally Shortest Paths (LSPs).** For a path  $\pi_{xy} \in G$ , we define the  $\pi_{xy}$  distance from  $x$  to  $y$  as  $\mathbf{w}(\pi_{xy}) = \sum_{e \in \pi_{xy}} \mathbf{w}(e)$ , and the  $\pi_{xy}$  length from  $x$  to  $y$  as the number of edges in  $\pi_{xy}$ . For any  $x, y \in V$ ,  $d(x, y)$  denotes the shortest path distance from  $x$  to  $y$  in  $G$ . A path  $\pi_{xy}$  in  $G$  is a *locally shortest path (LSP)* [3] if either  $\pi_{xy}$  contains a single vertex, or every proper subpath of  $\pi_{xy}$  is a shortest path in  $G$ . As noted in [3], every shortest path (SP) is an LSP, but an LSP need not be an SP (e.g., every single edge is an LSP).

The DI method maintains all LSPs in a graph with unique shortest paths, and these are key to efficiently maintaining shortest paths under decremental and fully dynamic updates. The decremental method we present here maintains all LSPs for all (multiple) shortest paths in a graph.

**Betweenness Centrality (BC).** Betweenness centrality is a widely-used measure in the analysis of large complex networks, and is defined as follows. For any pair  $x, y$  in  $V$ , let  $\sigma_{xy}$  denote the number of shortest paths from  $x$  to  $y$  in  $G$ , and let  $\sigma_{xy}(v)$  denote the number of shortest paths from  $x$  to  $y$  in  $G$  that pass through  $v$ . Then,  $BC(v) = \sum_{s \neq v, t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$ . This measure is often used as an index that determines the relative importance of  $v$  in the network. Some applications of BC include analyzing social interaction networks [7], identifying lethality in biological networks [11], and identifying key actors in terrorist networks [2, 8]. Heuristics for dynamic betweenness centrality with good experimental performance are given in [5, 9, 13], but none of these algorithms provably improve on the widely used static algorithm by Brandes [1], which runs in  $O(mn + n^2 \cdot \log n)$  time on any class of graphs, where  $m = |E|$ .

Recently, we gave a simple incremental BC algorithm [10], that provably improves on Brandes’ on sparse graphs, and also typically improves on Brandes’ in dense graphs (e.g., in the setting of Theorem 2 below). In this paper, we complement the results in [10]; however, decremental updates are considerably more challenging (similar to APSP, as noted in [3]).

The key step in the recent incremental BC algorithm [10] is the incremental maintenance of the APASP dags (achieved using techniques unrelated to the current paper). After the updated dags are obtained, the BC scores can be computed in time linear in the combined sizes of the APASP dags (plus  $O(n^2)$ ). Thus, if we instead use our decremental APASP algorithm in the key step in [10], we obtain a decremental algorithm for BC with the same bound as APASP.

**Our Results.** Let  $\nu^*$  be the maximum number of edges that lie on shortest paths through any given vertex in  $G$ ; thus,  $\nu^*$  also bounds the number of edges that lie on any single-source shortest path dag. Let  $m^*$  be the number of edges in  $G$  that lie on shortest paths (see, e.g., Karger et al. [6]). Our main result is the following theorem, where we have assumed that  $\nu^* = \Omega(n)$ .

**Theorem 1.** *Let  $\Sigma$  be a sequence of decremental updates on  $G = (V, E)$ . Then, all SP dags, all LSPs, and all BC scores can be maintained in amortized time  $O(\nu^{*2} \cdot \log n)$  per update when  $|\Sigma| = \Omega(m^*/\nu^*)$ .*

**Discussion of the Parameters.** As noted in [6], it is well-known that  $m^* = O(n \log n)$  with high probability in a complete graph where edge weights are chosen from a large class of probability distributions. Since  $\nu^* \leq m^*$ , our algorithms will have an amortized bound of  $O(n^2 \cdot \log^3 n)$  on such graphs. Also,  $\nu^* = O(n)$  in any graph with only a constant number of shortest paths between every pair of vertices, even though  $m^*$  can be  $\Theta(n^2)$  in the worst case even in graphs with unique shortest paths. In fact  $\nu^* = O(n)$  even in some graphs that have an exponential number of shortest paths between some pairs of vertices. In all such cases, and more generally, when the number of edges on shortest paths through any single vertex is  $O(n)$ , our algorithm will run in amortized  $O(n^2 \cdot \log n)$  time per decremental update. Thus we have:

**Theorem 2.** *Let  $\Sigma$  be a sequence of decremental updates on graphs where the number of edges on shortest paths through any single vertex is  $O(n)$ . Then, all SP dags, all LSPs, and all BC scores can be maintained in amortized time  $O(n^2 \cdot \log n)$  per update when  $|\Sigma| = \Omega(m^*/n)$ .*

**Corollary 1.** *If the number of shortest paths for any vertex pair is bounded by a constant, then decremental APASP, LSPs, and BC have amortized cost  $O(n^2 \cdot \log n)$  per update when the update sequence has length  $\Omega(m^*/n)$ .*

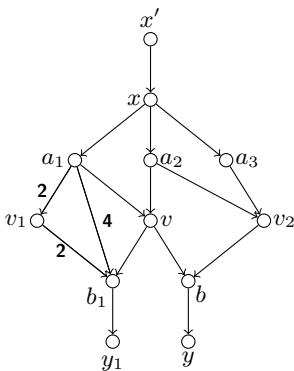


Fig. 1. Graph  $G$

Set	$G$ (before update on $v$ )
$P(x, y)$ $= P^*(x, y)$	$\{((xa_1, by), 4, 1), ((xa_2, by), 4, 2), ((xa_3, by), 4, 1)\}$
$P(x, b_1)$	$\{(xa_1, vb_1), 3, 1), ((xa_2, vb_1), 3, 1)\}$
$P^*(x, b_1)$	$\{((xa_1, vb_1), 3, 1), ((xa_2, vb_1), 3, 1)\}$
$L^*(v, y_1)$	$\{a_1, a_2\}$
$L(v, b_1y_1)$	$\{a_1, a_2\}$
$R^*(x, v)$	$\{b, b_1\}$
$R(xa_2, v)$	$\{b, b_1\}$

Fig. 2. A subset of the tuple-system for  $G$  in Fig. 1

**The DI Method.** Here we will use an example to give a quick review of the DI approach [3], which forms the basis for our method. Consider the graph  $G$  in Fig. 1, where all edges have weight 1 except for the ones with explicit weights.

As in DI, let us assume here that  $G$  has been pre-processed to identify a unique shortest path between every pair of vertices. In  $G$  the shortest path from  $a_1$  to  $b_1$  is  $\langle a_1, v, b_1 \rangle$  and has weight 2, and by definition, the paths  $p_1 = \langle a_1, b_1 \rangle$  and  $p_2 = \langle a_1, v_1, b_1 \rangle$  of weight 4 are both LSPs. Now consider a decremental update on  $v$  that increases  $\mathbf{w}(a_1, v)$  to 10 and  $\mathbf{w}(a_2, v)$  to 5, and let  $G'$  be the resulting graph (see Fig. 3). In  $G'$  both  $p_1$  and  $p_2$  become shortest paths. Furthermore, a *left extension* of the path  $p_1$ , namely  $p_3 = \langle x, a_1, b_1 \rangle$  becomes a shortest path from  $x$  to  $b_1$  in  $G'$ . Note that the path  $p_3$  is not even an LSP in the graph  $G$ ; however, it is obtained as a left extension of a path that has become shortest after the update.

The elegant method of storing LSPs and creating longer LSPs by left and right extending shortest paths is the basis of the DI approach [3]. To achieve this, the DI approach uses a succinct representation of SPs, LSPs and their left and right extensions using suitable data structures. It then uses a procedure *cleanup* to remove from the data structures all the shortest paths and LSPs that contain the updated vertex  $v$ , and a complementary procedure *fixup* that first adds all the trivial LSPs (corresponding to edges incident on  $v$ ), and then restores the shortest paths and LSPs between all pairs of vertices. The DI approach thus efficiently maintains a single shortest path between all pairs of vertices under decremental updates.

In this paper we are interested in maintaining *all* shortest paths for all vertex pairs and this requires several enhancements to the methods in [3]. In Section 2 we present a new *tuple system* which succinctly represents all LSPs in a graph with multiple shortest paths. In the rest of the paper we present our decremental algorithm for maintaining this tuple system, and hence for maintaining APASP and BC scores.

## 2 A System of Tuples

In this section we present an efficient representation of the set of SPs and LSPs for an edge weighted graph  $G = (V, E)$ . We first define the notions of *tuple* and *triple*.

**Tuple.** A tuple,  $\tau = (xa, by)$ , represents the set of LSPs in  $G$ , all of which use the same first edge  $(x, a)$  and the same last edge  $(b, y)$ . The weight of every path represented by  $\tau$  is  $\mathbf{w}(x, a) + d(a, b) + \mathbf{w}(b, y)$ . We call  $\tau$  a *locally shortest path tuple (LST)*. In addition, if  $d(x, y) = \mathbf{w}(x, a) + d(a, b) + \mathbf{w}(b, y)$ , then  $\tau$  is a *shortest path tuple (ST)*. Fig. 5(a) shows a tuple  $\tau$ .

**Triple.** A triple  $\gamma = (\tau, wt, count)$ , represents the tuple  $\tau = (xa, by)$  that contains  $count > 0$  number of paths from  $x$  to  $y$ , each with weight  $wt$ . In Fig. 1, the triple  $((xa_2, by), 4, 2)$  represents two paths from  $x$  to  $y$ , namely  $p_1 = \langle x, a_2, v, b, y \rangle$  and  $p_2 = \langle x, a_2, v_2, b, y \rangle$  both having weight 4.

**Storing Locally Shortest Paths.** We use triples to succinctly store all LSPs and SPs for each vertex pair in  $G$ . For  $x, y \in V$ , we define:

$$P(x, y) = \{((xa, by), wt, count): (xa, by) \text{ is an LST from } x \text{ to } y \text{ in } G\}$$

$$P^*(x, y) = \{((xa, by), wt, count): (xa, by) \text{ is an ST from } x \text{ to } y \text{ in } G\}.$$

Note that all triples in  $P^*(x, y)$  have the same weight. We will use the term LST to denote either a locally shortest tuple or a triple representing a set of LSPs, and it will be clear from the context whether we mean a triple or a tuple.

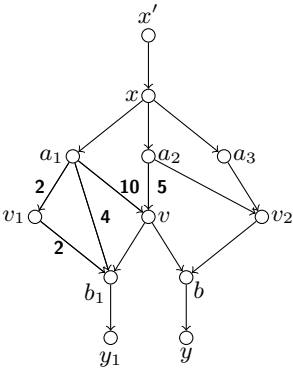


Fig. 3. Graph  $G'$

Set	$G'$ (with $w(a_1, v) = 10, w(a_2, v) = 5$ )
$P(x, y)$ $= P^*(x, y)$	$\{((xa_2, by), 4, 1), ((xa_3, by), 4, 1)\}$
$P(x, b_1)$	$\{((xa_1, v_1b_1), 5, 1), ((xa_2, vb_1), 7, 1), ((xa_1, a_1b_1), 5, 1)\}$
$P^*(x, b_1)$	$\{((xa_1, v_1b_1), 5, 1), ((xa_1, a_1b_1), 5, 1)\}$
$L^*(v, y_1)$	$\{a_2\}$
$L(v, b_1y_1)$	$\{a_2\}$
$R^*(x, v)$	$\emptyset$
$R(xa_2, v)$	$\{b_1\}$

Fig. 4. A subset of the tuple-system for  $G'$

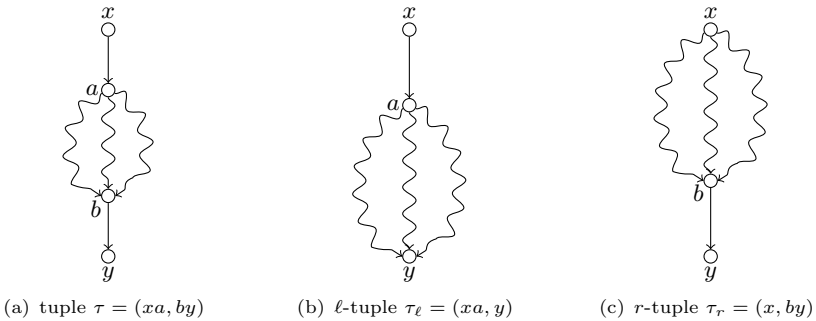


Fig. 5. Tuples

**Left Tuple and Right Tuple.** A left tuple (or  $\ell$ -tuple),  $\tau_\ell = (xa, y)$ , represents the set of LSPs from  $x$  to  $y$ , all of which use the same first edge  $(x, a)$ . The weight of every path represented by  $\tau_\ell$  is  $w(x, a) + d(a, y)$ . If  $d(x, y) = w(x, a) + d(a, y)$ , then  $\tau_\ell$  represents the set of shortest paths from  $x$  to  $y$ , all of which use the first edge  $(x, a)$ . A right tuple ( $r$ -tuple)  $\tau_r = (x, by)$  is defined analogously. Fig. 5(b)

and Fig. 5(c) show a left tuple and a right tuple respectively. In the following, we will say that a tuple (or  $\ell$ -tuple or  $r$ -tuple) *contains* a vertex  $v$ , if at least one of the paths represented by the tuple contains  $v$ .

**ST and LST Extensions.** For a shortest path  $r$ -tuple  $\tau_r = (x, by)$ , we define  $L(\tau_r)$  to be the set of vertices which can be used as pre-extensions to create LSTs in  $G$ . Similarly, for a shortest path  $\ell$ -tuple  $\tau_\ell = (xa, y)$ ,  $R(\tau_\ell)$  is the set of vertices which can be used as post-extensions to create LSTs in  $G$ . We do not define  $R(\tau_r)$  and  $L(\tau_\ell)$ . So we have:

$$L(x, by) = \{x' : (x', x) \in E(G) \text{ and } (x'x, by) \text{ is an LST in } G\}$$

$$R(xa, y) = \{y' : (y, y') \in E(G) \text{ and } (xa, yy') \text{ is an LST in } G\}.$$

For  $x, y \in V$ ,  $L^*(x, y)$  denotes the set of vertices which can be used as pre-extensions to create shortest path tuples in  $G$ ;  $R^*(x, y)$  is defined symmetrically:

$$L^*(x, y) = \{x' : (x', x) \in E(G) \text{ and } (x'x, y) \text{ is a } \ell\text{-tuple representing SPs in } G\}$$

$$R^*(x, y) = \{y' : (y, y') \in E(G) \text{ and } (x, yy') \text{ is an } r\text{-tuple representing SPs in } G\}.$$

Fig. 2 shows a subset of these sets for the graph  $G$  in Fig. 1.

**Key Deviations from DI [3].** The assumption of unique shortest paths in [3] ensures that  $\tau = (xa, by)$ ,  $\tau_\ell = (xa, y)$ , and  $\tau_r = (x, by)$  all represent exactly the same (single) locally shortest path. However, in our case, the set of paths represented by  $\tau_\ell$  and  $\tau_r$  can be different, and  $\tau$  is a subset of paths represented by  $\tau_\ell$  and  $\tau_r$ . Our definitions of ST and LST extensions are derived from the analogous definitions in [3] for SP and LSP extensions of paths. For a path  $\pi = x \rightarrow a \rightsquigarrow b \rightarrow y$ , DI defines sets  $L$ ,  $L^*$ ,  $R$  and  $R^*$ . In our case, the analog of a path  $\pi = x \rightarrow a \rightsquigarrow b \rightarrow y$  is a tuple  $\tau = (xa, by)$ , but to obtain efficiency, we define the set  $L$  only for an  $r$ -tuple and the set  $R$  only for an  $\ell$ -tuple. Furthermore, we define  $L^*$  and  $R^*$  for each pair of vertices.

In the following two lemmas we bound the total number of tuples in the graph and the total number of tuples that contain a given vertex  $v$ . These bounds also apply to the number of triples since there is exactly one triple for each tuple in our tuple system.

**Lemma 1.** *The number of LSTs in  $G = (V, E)$  is bounded by  $O(m^* \cdot \nu^*)$ .*

*Proof.* For any LST  $(\times a, \times \times)$ , for some  $a \in V$ , the first and last edge of any such tuple must lie on a shortest path containing  $a$ . Let  $E_a^*$  denote the set of edges that lie on shortest paths through  $a$ , and let  $I_a$  be the set of incoming edges to  $a$ . Then, there are at most  $\nu^*$  ways of choosing the last edge in  $(\times a, \times \times)$  and at most  $E_a^* \cap I_a$  ways of choosing the first edge in  $(\times a, \times \times)$ . Since  $\sum_{a \in V} |E_a^* \cap I_a| = m^*$ , the number of LSTs in  $G$  is at most  $\sum_{a \in V} \nu^* \cdot |E_a^* \cap I_a| \leq m^* \cdot \nu^*$ .  $\square$

**Lemma 2.** *The number of LSTs that contain a vertex  $v$  is  $O(\nu^{*2})$ .*

*Proof.* We distinguish three different cases:

1. Tuples starting with  $v$ : for a tuple that starts with edge  $(v, a)$ , the last edge must lie on  $a$ 's SP dag, so there are at most  $\nu^*$  choices for the last edge. Hence, the number of tuples with  $v$  as start vertex is at most  $\sum_{a \in V \setminus v} \nu^* \leq n \cdot \nu^*$ .
2. Similarly, the number of tuples with  $v$  as end vertex is at most  $n \cdot \nu^*$ .
3. For any tuple  $\tau = (xa, by)$  that contains  $v$  as an internal vertex, both  $(x, a)$  and  $(b, y)$  lie on a shortest path through  $v$ , hence the number of such tuples is at most  $\nu^{*2}$ . □

### 3 Decremental Algorithm

Here we present our decremental APASP algorithm. Recall that a decremental update on a vertex  $v$  either deletes or increases the weights of a subset of edges incident on  $v$ . We begin with the data structures we use.

**Data Structures.** For every  $x, y, x \neq y$  in  $V$ , we maintain the following:

1.  $P(x, y)$  – a priority queue containing LSTs from  $x$  to  $y$  with weight as key.
2.  $P^*(x, y)$  – a priority queue containing STs from  $x$  to  $y$  with weight as key.
3.  $L^*(x, y)$  – a balanced search tree containing vertices with vertex ID as key.
4.  $R^*(x, y)$  – a balanced search tree containing vertices with vertex ID as key.

For every  $\ell$ -tuple we have its right extension, and for every  $r$ -tuple its left extension. These sets are stored as balanced search trees (BSTs) with the vertex ID as a key. Additionally, we maintain all tuples in a BST *dict*, with a tuple  $\tau = (xa, by)$  having key  $[x, y, a, b]$ . We also maintain pointers from  $\tau$  to  $R(xa, y)$  and  $L(x, by)$ , and to the corresponding triple containing  $\tau$  in  $P(x, y)$ , (and in  $P^*(x, y)$  if  $(xa, by)$  is an ST). Finally, we maintain a sub-dictionary of *dict* called Marked-Tuples (explained below). Marked-Tuples, unlike the other data structures, is specific only to one update.

**The Algorithm.** Given the updated vertex  $v$  and the updated weight function  $\mathbf{w}'$  over all the incoming and outgoing edges of  $v$ , the decremental algorithm performs two main steps *cleanup* and *fixup*, as in DI. The cleanup procedure removes from the tuple system every LSP that contains the updated vertex  $v$ . The following definition of a *new* LSP is from DI [3].

**Definition 1.** *A path that is shortest (locally shortest) after an update to vertex  $v$  is new if either it was not an SP (LSP) before the update, or it contains  $v$ .*

The fixup procedure adds to the tuple system all the *new* shortest and locally shortest paths. In contrast to DI, recall that we store locally shortest paths in  $P$  and  $P^*$  as triples. Hence removing or adding paths implies decrementing or incrementing the count in the relevant triple; thus a triple is removed or added only if its count goes down to zero or up from zero. Moreover, new tuples may be created through combining several existing tuples. Some of the updated data structures for the graph  $G'$  in Fig. 3, obtained after a decremental update on  $v$  in the graph  $G$  in Fig. 1, are schematized in Fig. 4.



### 3.1 The Cleanup Procedure

Alg. 1 (cleanup) uses an initially empty heap  $H_c$  of triples. It also initializes the empty dictionary Marked-Tuples. The algorithm then creates the trivial triple corresponding to the vertex  $v$  and adds it to  $H_c$  (Step 2, Alg. 1). For a triple  $((xa, by), wt, count)$  the key in  $H_c$  is  $[wt, x, y]$ . The algorithm repeatedly extracts min-key triples from  $H_c$  (Step 4, Alg. 1) and processes them. The processing of triples involves left-extending (Steps 5–17, Alg. 1) and right-extending triples (Step 18, Alg. 1) and removing from the tuple system the set of LSPs thus formed. This is similar to cleanup in DI. However, since we deal with a set of paths instead of a single path, we need significant modifications, of which we now highlight two: (i) Accumulation used in Step 4 and (ii) use of Marked-Tuples in Step 7 and Step 11.

---

**Algorithm 1.** cleanup( $v$ )

---

```

1:  $H_c \leftarrow \emptyset$ ; Marked-Tuples  $\leftarrow \emptyset$ 
2:  $\gamma \leftarrow ((vv, vv), 0, 1)$ ; add  $\gamma$  to  $H_c$ 
3: while  $H_c \neq \emptyset$  do
4:   extract in  $S$  all the triples with min-key  $[wt, x, y]$  from  $H_c$ 
5:   for every  $b$  such that  $(x \times, by) \in S$  do
6:     let  $fcoun't' = \sum_i ct_i$  such that  $((xa_i, by), wt, ct_i) \in S$ 
7:     for every  $x' \in L(x, by)$  such that  $(x'x, by) \notin$  Marked-Tuples do
8:        $wt' \leftarrow wt + \mathbf{w}(x', x)$ ;  $\gamma' \leftarrow ((x'x, by), wt', fcoun't')$ ; add  $\gamma'$  to  $H_c$ 
9:       remove  $\gamma'$  in  $P(x', y)$  // decrements count by fcoun't'
10:      if a triple for  $(x'x, by)$  exists in  $P(x', y)$  then
11:        insert  $(x'x, by)$  in Marked-Tuples
12:      else
13:        delete  $x'$  from  $L(x, by)$  and delete  $y$  from  $R(x'x, b)$ 
14:        if a triple for  $(x'x, by)$  exists in  $P^*(x', y)$  then
15:          remove  $\gamma'$  in  $P^*(x', y)$  // decrements count by fcoun't'
16:          if  $P^*(x, y) = \emptyset$  then delete  $x'$  from  $L^*(x, y)$ 
17:          if  $P^*(x', b) = \emptyset$  then delete  $y$  from  $R^*(x', b)$ 
18:      perform symmetric steps 5 – 17 for right extensions

```

---

**Accumulation.** In Step 4 we extract a collection  $S$  of triples all with key  $[wt, x, y]$  from  $H_c$  and process them together in that iteration of the while loop. Assume that for a fixed last edge  $(b, y)$ ,  $S$  contains triples of the form  $(xa_t, by)$ , for  $t = 1, \dots, k$ . Our algorithm processes and left-extends all these triples with the same last edge together. This ensures that, for any  $x' \in L(x, by)$ , we generate the triple  $(x'x, by)$  exactly once. The accumulation is correct because any valid left extension for a triple  $(xa_i, by)$  is also a valid left extension for  $(xa_j, by)$  when both triples have the same weight.

**Marked-Tuples.** The dictionary of Marked-Tuples is used to ensure that every path through the vertex  $v$  is removed from the tuple system exactly once and therefore counts of paths in triples are correctly maintained. Note that a path of the form  $(xa, by)$  can be generated either as a left extension of  $(a, by)$  or by a right extension of  $(xa, b)$ . This is true in DI as well. However, due to the assumption of unique shortest paths they do not need to maintain counts of paths, and hence do not require the book-keeping using Marked-Tuples.

**3.1.1 Complexity and Correctness.** Lemma 3 establishes the correctness of Alg. 1 and can be proved using a suitable loop invariant for the while loop in Step 3. The time bound in Lemma 4 follows from Lemma 2 since every triple examined in cleanup has at least one path that contains  $v$ .

**Lemma 3.** *After Alg. 1 is executed, the counts of triples in  $P$  ( $P^*$ ) represent counts of LSPs (SPs) in  $G$  that do not pass through  $v$ . Moreover, the sets  $L, L^*, R, R^*$  are correctly maintained.*

**Lemma 4.** *For an update on a vertex  $v$ , Alg. 1 takes  $O(v^{*2} \cdot \log n)$  time.*

## 3.2 The Fixup Procedure

The goal of the fixup procedure is to add to the tuple system all *new* shortest and locally shortest paths (recall Definition 1).

The fixup procedure (pseudo-code in Alg. 2) works with a heap of triples ( $H_f$  here), which is initialized with a *candidate* shortest path triple for each pair of vertices. The algorithm repeatedly extracts the set of triples with minimum key and processes them. The main invariant for the algorithm (similar to DI [3]) is that for a pair  $x, y$ , the weight of the first set of triples extracted from  $H_f$  gives the distance from  $x$  to  $y$  in the updated graph. Thus, these triples are all identified as shortest path triples, and we need to extend them if in fact they represent *new* shortest paths. To readily identify triples containing paths through  $v$  we use some additional book-keeping: for every triple  $\gamma$  we store the update number ( $\text{update-num}(\gamma)$ ) and a count of the number of paths in that triple that pass through  $v$  ( $\text{paths}(\gamma, v)$ ). Finally, similar to cleanup, the fixup procedure also left and right extends triples to create triples representing new locally shortest paths.

Alg. 2 initializes  $H_f$  in Steps 2–5 as follows. (i) For every edge incident on  $v$ , it creates a trivial triple  $\gamma$  which is inserted into  $H_f$  and  $P$ . It also sets  $\text{update-num}(\gamma)$  and  $\text{paths}(\gamma, v)$  for each such  $\gamma$ ; (ii) For every  $x, y \in V$ , it adds a candidate min-weight triple from  $P(x, y)$  to  $H_f$  (even if  $P(x, y)$  contains several min-weight triples; this is done for efficiency).

Alg. 2 executes Steps 10–17 when for a pair  $x, y$ , the first set of triples  $S'$ , all of weight  $wt$ , are extracted from  $H_f$ . We claim (Invariant 3) that  $wt$  denotes the shortest path distance from  $x$  to  $y$  in the updated graph. The goal of Steps 10–17 is to create a set  $S$  of triples that represent *new* shortest paths, and this step is considerably more involved than the corresponding step in DI. In DI [3], only a single path  $p$  is extracted from  $H_f$  possibly resulting in a *new* shortest path from  $x$  to  $y$ . If  $p$  is *new* then it is added to  $P^*$  and the algorithm extends it to create new LSP. In our case, we extract not just multiple paths but multiple shortest path triples from  $x$  to  $y$ , and some of these triples may not be in  $H_f$ . We now describe how our algorithm generates the new shortest paths in Steps 10–17.

Steps 10–17, Alg. 2 – As mentioned above, Steps 10–17 create a set  $S$  of triples that represent *new* shortest paths. There are two cases.

- $P^*(x, y)$  is empty: Here, we process the triples in  $S'$ , but in addition, we may be required to process triples of weight  $wt$  from the set  $P(x, y)$ . To see this, consider the example in Fig. 1 and consider the pair  $a_1, b_1$ . In  $G$ , there is one shortest path  $\langle a_1, v, b_1 \rangle$  which is removed from  $P(a_1, b_1)$  and  $P^*(a_1, b_1)$  during cleanup. In  $G'$ ,  $d(a_1, b_1) = 4$  and there are 2 shortest paths, namely  $p_1 = \langle a_1, b_1 \rangle$  and  $p_2 = \langle a_1, v_1, b_1 \rangle$ . Note that both of these are LSPs in  $G$  and therefore are present in  $P(a_1, b_1)$ . In Step 5, Alg. 2 we insert exactly one of them into the heap  $H_f$ . However, both need to be processed and also left and right extended to create new locally shortest paths. Thus, under this condition, we examine all the min-weight triples present in  $P(a_1, b_1)$ .
- $P^*(x, y)$  is non-empty: After a decremental update, the distance from  $x$  to  $y$  can either remain the same or increase, but it cannot decrease. Further, cleanup removed from the tuple system all paths that contain  $v$ . Hence, if  $P^*(x, y)$  is non-empty at this point, it implies that all paths in  $P^*(x, y)$  avoid  $v$ . In this case, we can show (Invariant 4) that it suffices to only examine the triples present in  $H_f$ . Furthermore, the only paths that we need to process are the paths that pass through the vertex  $v$ .

Steps 19–29, Alg. 2 – These steps left-extend and right-extend the triples in  $S$  representing *new* shortest paths from  $x$  to  $y$ .

---

**Algorithm 2.** `fixup( $v, w'$ )`

---

```

1:  $H_f \leftarrow \emptyset$ ; Marked-Tuples  $\leftarrow \emptyset$ 
2: for each edge incident on  $v$  do
3:   create a triple  $\gamma$ ; set  $paths(\gamma, v) = 1$ ; set  $update\_num(\gamma)$ ; add  $\gamma$  to  $H_f$  and to  $P()$ 
4: for each  $x, y \in V$  do
5:   add a min-weight triple from  $P(x, y)$  to  $H_f$ 
6: while  $H_f \neq \emptyset$  do
7:   extract in  $S'$  all triples with min-key  $[wt, x, y]$  from  $H_f$ ;  $S \leftarrow \emptyset$ 
8:   if  $S'$  is the first extracted set from  $H_f$  for  $x, y$  then
9:     {Steps 10–17: add new STs (or increase counts of existing STs) from  $x$  to  $y$ .}
10:    if  $P^*(x, y)$  is empty then
11:      for each  $\gamma' \in P(x, y)$  with weight  $wt$  do
12:        let  $\gamma' = ((x a', b' y), wt, count')$ 
13:        add  $\gamma'$  to  $P^*(x, y)$  and  $S$ ; add  $x$  to  $L^*(a', y)$  and  $y$  to  $R^*(x, b')$ 
14:    else
15:      for each  $\gamma' \in S'$  containing a path through  $v$  do
16:        let  $\gamma' = ((x a', b' y), wt, count')$ 
17:        add  $\gamma'$  with  $paths(\gamma', v)$  in  $P^*(x, y)$  and  $S$ ; add  $x$  to  $L^*(a', y)$  and  $y$  to  $R^*(x, b')$ 
18:    {Steps 19–28: add new LSTs (or increase counts of existing LSTs) that extend SPs from  $x$  to  $y$ .}
19:    for every  $b$  such that  $(x \times, by) \in S$  do
20:      let  $fcount' = \sum_i ct_i$  such that  $((x a_i, by), wt, ct_i) \in S$ 
21:      for every  $x'$  in  $L^*(x, b)$  do
22:        if  $(x' x, by) \notin$  Marked-Tuples then
23:           $wt' \leftarrow wt + w(x', x)$ ;  $\gamma' \leftarrow ((x' x, by), wt', fcount')$ 
24:          set  $update\_num(\gamma')$ ;  $paths(\gamma', v) \leftarrow \sum_{\gamma=(x \times, by)} paths(\gamma, v)$ ; add  $\gamma'$  to  $H_f$ 
25:          if a triple for  $(x' x, by)$  exists in  $P(x', y)$  then
26:            add  $\gamma'$  with  $paths(\gamma', v)$  in  $P(x', y)$ ; add  $(x' x, by)$  to Marked-Tuples
27:          else
28:            add  $\gamma'$  to  $P(x', y)$ ; add  $x'$  to  $L(x, by)$  and  $y$  to  $R(x' x, b)$ 
29:          perform steps symmetric to Steps 19 – 28 for right extensions.

```

---

Fixup maintains the following invariants. Invariant 3 is proved similarly to Invariant 3.1 in [3]. The proof of Invariant 4 requires a careful analysis of various cases to show that indeed all *new* shortest paths are inserted into the set  $S$ .

Finally, Lemma 5 is proved using a suitable loop invariant for the while loop in Step 6 of Alg. 2.

**Invariant 3.** *If the set  $S'$  in Step 7 of Alg. 2 is the first extracted set from  $H_f$  for  $x, y$ , then the weight of each triple in  $S'$  is the shortest path distance from  $x$  to  $y$  in the updated graph.*

**Invariant 4.** *The set  $S$  of triples constructed in Steps 10–17 of Alg. 2 represents all of the new shortest paths from  $x$  to  $y$ .*

**Lemma 5.** *After execution of Alg. 2, for any  $(x, y) \in V$ , the counts of the triples in  $P(x, y)$  and  $P^*(x, y)$  represent the counts of LSPs and SPs from  $x$  to  $y$  in the updated graph. Moreover, the sets  $L, L^*, R, R^*$  are correctly maintained.*

**3.2.1 Complexity of Fixup.** As in DI, we observe that shortest paths and LSPs are removed only in cleanup and are added only in fixup. In a call to fixup, accessing a triple takes  $O(\log n)$  time since it is accessed on a constant number of data structures. So, it suffices to bound the number of triples accessed in a call to fixup, and then multiply that bound by  $O(\log n)$ .

We will establish an amortized bound. The total number of LSTs at any time, including the end of the update sequence, is  $O(m^* \cdot \nu^*)$  (by Lemma 1). Hence, if fixup accessed only *new* triples outside of the  $O(n^2)$  triples added initially to  $H_f$ , the amortized cost of fixup (for a long enough update sequence) would be  $O(\nu^{*2} \cdot \log n)$ , the cost of a cleanup. This is in fact the analysis in DI, where fixup satisfies this property. However, in our algorithm fixup accesses several triples that are already in the tuple system: In Steps 11–13 we examine triples already in  $P$ , in Steps 15–17 we could increment the count of an existing triple in  $P^*$ , and in Steps 19–28 we increment the count of an existing triple in  $P$ . We bound the costs of these steps in Lemma 6 below by classifying each triple  $\gamma$  as one of the following disjoint types:

- **Type-0 (contains- $v$ ):**  $\gamma$  represents at least one path containing vertex  $v$ .
- **Type-1 (new-LST):**  $\gamma$  was not an LST before the update but is an LST after the update, and no path in  $\gamma$  contains  $v$ .
- **Type-2 (new-ST-old-LST):**  $\gamma$  is an ST after the update, and  $\gamma$  was an LST but not an ST before the update, and no path in  $\gamma$  contains  $v$ .
- **Type-3 (new-ST-old-ST):**  $\gamma$  was an ST before the update and continues to be an ST after the update, and no path in  $\gamma$  contains  $v$ .
- **Type-4 (new-LST-old-LST):**  $\gamma$  was an LST before the update and continues to be an LST after the update, and no path in  $\gamma$  contains  $v$ .

The following lemma establishes an amortized bound for fixup which is the same as the worst case bound for cleanup. This proves Theorem 1.

**Lemma 6.** *The fixup procedure takes time  $O(\nu^{*2} \cdot \log n)$  amortized over a sequence of  $\Omega(m^*/\nu^*)$  decremental-only updates.*

*Proof.* We bound the number of triples examined; the time taken is  $O(\log n)$  times the number of triples examined due to the data structure operations performed on a triple. The initialization in Steps 1–5 takes  $O(n^2)$  time. We now consider the triples examined after Step 5. The number of Type-0 triples is  $O(\nu^{*2})$  by Lemma 2. The number of Type-1 triples is addressed by amortizing over the entire update sequence as described in the paragraph below. For Type-2 triples we observe that since updates only increase the weights on edges, a shortest path never reverts to being an LSP. Further, each such Type-2 triple is examined only a constant number of times (in Steps 10–13). Hence we charge each access to a Type-2 triple to the step in which it was created as a Type-1 triple. For Type-3 and Type-4, we note that for any  $x, y$  we add exactly one candidate min-weight triple from  $P(x, y)$  to  $H_f$ , hence initially there are at most  $n^2$  such triples in  $H_f$ . Moreover, we never process an old LST which is not an ST so no additional Type-4 triples are examined during fixup. Finally, triples in  $P^*$  that are not placed initially in  $H_f$  are not examined in any step of fixup, so no additional Type-3 triples are examined. Thus the number of triples examined by a call to fixup is  $O(\nu^{*2})$  plus  $O(X)$ , where  $X$  is the number of *new* triples fixup adds to the tuple system. (This includes an  $O(1)$  credit placed on each new LST for a possible later conversion to an ST.)

Let  $\sigma$  be the number of updates in the update sequence. Since triples are removed only in cleanup, at most  $O(\sigma \cdot \nu^{*2})$  triples are removed by the cleanups. There can be at most  $O(m^* \cdot \nu^*)$  triples remaining at the end of the sequence (by Lemma 1), hence the total number of new triples added by all fixups in the update sequence is  $O(\sigma \cdot \nu^{*2} + m^* \cdot \nu^*)$ . When  $\sigma > m^*/\nu^*$ , the first term dominates, and this gives an average of  $O(\nu^{*2})$  triples added per fixup, and the desired amortized time bound for fixup.  $\square$

**Discussion.** We have presented an efficient decremental algorithm to maintain all-pairs all shortest paths (APASP). The space used by our algorithm is  $O(m^* \cdot \nu^*)$ , the worst case number of triples in our tuple system. By using this decremental APASP algorithm in place of the incremental APASP algorithm used in [10], we obtain a decremental algorithm for maintaining BC scores with the same bound.

Very recently, two of the authors have obtained a fully dynamic APASP algorithm [12] that combines elements in the fully dynamic APSP algorithms in [3] and [14], while building on the results in the current paper. When specialized to unique shortest paths (i.e., APSP), this algorithm is about as simple as the one in [3] and matches its amortized bound.

## References

1. Brandes, U.: A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology* **25**(2), 163–177 (2001)
2. Coffman, T., Greenblatt, S., Marcus, S.: Graph-based technologies for intelligence analysis. *Commun. ACM* **47**(3), 45–47 (2004)

3. Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. *J. ACM* **51**(6), 968–992 (2004)
4. Freeman, L.C.: A set of measures of centrality based on betweenness. *Sociometry* **40**(1), 35–41 (1977)
5. Green, O., McColl, R., Bader, D.A.: A fast algorithm for streaming betweenness centrality. In: *Proc. of 4th PASSAT*, pp. 11–20 (2012)
6. Karger, D.R., Koller, D., Phillips, S.J.: Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM J. Comput.* **22**(6), 1199–1217 (1993)
7. Kourtellis, N., Alahakoon, T., Simha, R., Iamnitchi, A., Tripathi, R.: Identifying high betweenness centrality nodes in large social networks. In: *Social Network Analysis and Mining*, pp. 1–16 (2012)
8. Krebs, V.: Mapping networks of terrorist cells. *CONNECTIONS* **24**(3), 43–52 (2002)
9. Lee, M.-J., Lee, J., Park, J.Y., Choi, R.H., Chung, C.-W.: Qube: a quick algorithm for updating betweenness centrality. In: *Proc. of 21st WWW*, pp. 351–360 (2012)
10. Nasre, M., Pontecorvi, M., Ramachandran, V.: Betweenness Centrality – Incremental and Faster. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) *MFCS 2014, Part II*. LNCS, vol. 8635, pp. 577–588. Springer, Heidelberg (2014)
11. Pinney, J.W., McConkey, G.A., Westhead, D.R.: Decomposition of biological networks using betweenness centrality. In: *Proc. of RECOMB* (2005)
12. Pontecorvi, M., Ramachandran, V.: Fully dynamic all pairs all shortest paths and betweenness centrality. (2014) (manuscript)
13. Goel, K., Singh, R.R., Iyengar, S., Sukrit, : A Faster Algorithm to Update Betweenness Centrality after Node Alteration. In: Bonato, A., Mitzenmacher, M., Prałat, P. (eds.) *WAW 2013*. LNCS, vol. 8305, pp. 170–184. Springer, Heidelberg (2013)
14. Thorup, M.: Fully-Dynamic All-Pairs Shortest Paths: Faster and Allowing Negative Cycles. In: Hagerup, T., Katajainen, J. (eds.) *SWAT 2004*. LNCS, vol. 3111, pp. 384–396. Springer, Heidelberg (2004)

## Author Index

- Abu-Khizam, Faisal N. 479  
Aichholzer, Oswin 15  
Ando, Ei 376  
Angel, Eric 247  
Arimura, Hiroki 94  
Arulselvan, Ashwin 741  
Asano, Tetsuo 553  
Aurora, Pawan 674
- Bampis, Evripidis 247, 259  
Bazgan, Cristina 479  
Becker, Ruben 753  
Biedl, Therese 117  
Bohler, Cecilia 27  
Borodin, Allan 528  
Bose, Presenjit 181  
Bose, Prosenjit 313  
Bulteau, Laurent 298
- Cardinal, Jean 15  
Chao, Kun-Mao 606  
Chau, Vincent 247  
Cheng, Zhanpeng 581  
Chopin, Morgan 479  
Crochemore, Maxime 220
- Da Lozzo, Giordano 633  
Dahlgaard, Søren 141  
Demaine, Erik D. 389  
Demaine, Martin L. 389  
Domañiç, Nevzat Onur 646  
Douïeb, Karim 181  
Drange, Pål Grønås 285  
Dregi, Markus Sortland 285  
Droschinsky, Andre 81
- El-Zein, Hicham 543  
Eppstein, David 581
- Felsner, Stefan 726  
Feng, Haodi 491  
Fernau, Henning 479  
Floderus, Peter 65
- Fong, Ken 41  
Fox-Epstein, Eli 389  
Froese, Vincent 298
- Gajarský, Jakub 441  
Gavruskin, Alexander 235  
Gawrychowski, Paweł 701  
Ghosal, Pratik 593  
Groß, Martin 741  
Gutowski, Grzegorz 507  
Gąsieniec, Leszek 53
- Hartung, Sepp 298  
He, Meng 128, 565  
Heinemann, Bernhard 81  
Henze, Matthias 714  
Hliněný, Petr 441  
Hoang, Duc A. 389  
Huang, Norman 528  
Huber, Stefan 117
- Iacono, John 181  
Iliopoulos, Costas S. 220  
Inoue, Yuma 103  
Ito, Takehiro 195, 208, 389  
Izumi, Taisuke 553
- Jansson, Jesper 65, 414  
Jaume, Rafel 714  
Jelínek, Vít 633  
Jiang, Haitao 491
- Kakimura, Naonori 195  
Kamiyama, Naoyuki 195  
Kamiński, Marcin 208  
Kane, Daniel M. 273  
Karrenbauer, Andreas 753  
Khousainov, Bakhadyr 235  
Kijima, Shuji 376  
Kiyomi, Masashi 553  
Klavík, Pavel 401  
Knudsen, Mathias Bæk Tejs 141  
Kobayashi, Yusuke 195

- Kociumaka, Tomasz 220  
 Kodama, Chikaaki 365  
 Kohira, Yukihide 365  
 Kokho, Mikhail 235  
 Kolev, Pavel 621  
 Konagaya, Matsuo 553  
 Kozik, Jakub 507  
 Kratochvíl, Jan 633  
 Kriege, Nils 81  
 Kusters, Vincent 15
- Langerman, Stefan 15, 181  
 Lehre, Per Kristian 686  
 Letsios, Dimitrios 259  
 Levcopoulos, Christos 53, 65  
 Li, Jian 326  
 Li, Minming 41  
 Li, Xingfu 467  
 Liang, Hongyu 41  
 Lin, Guohui 353  
 Lin, Wei-Yin 606  
 Lingas, Andrzej 53, 65  
 Liu, Chih-Hung 27  
 Liu, Jiamou 235  
 Lucarelli, Giorgio 259
- Matsui, Tomomi 365  
 Mehta, Shashank K. 674  
 Micek, Piotr 507, 516  
 Minato, Shin-ichi 103  
 Morin, Pat 313  
 Mouawad, Amer E. 452  
 Munro, J. Ian 169, 543, 565  
 Mutzel, Petra 81
- Nagamochi, Hiroshi 429  
 Nasre, Meghana 593, 766  
 Navarro, Gonzalo 169  
 Niedermeier, Rolf 298  
 Nielsen, Jesper Sindahl 169  
 Nimbhorkar, Prajakta 593  
 Nishimura, Naomi 452
- Obdržálek, Jan 441  
 Okamoto, Yoshio 195  
 Ono, Hirotaka 208, 389, 553  
 Ordyniak, Sebastian 441  
 Otachi, Yota 389, 553
- Palfrader, Peter 117  
 Papadopoulou, Evanthia 27  
 Pilz, Alexander 726  
 Plaxton, C. Gregory 646  
 Policriti, Alberto 157  
 Pontecorvi, Matteo 766  
 Prezza, Nicola 157
- Radoszewski, Jakub 220  
 Ramachandran, Vijaya 766  
 Raman, Venkatesh 452, 543  
 Rotbart, Noy 141  
 Rusak, Damian 701  
 Rutter, Ignaz 633  
 Rytter, Wojciech 220
- Saumell, Maria 401  
 Schweitzer, Pascal 553  
 Shah, Rahul 169  
 Skutella, Martin 741  
 Sledneu, Dzmitry 65  
 Sun, He 621  
 Sun, Xiaoming 661  
 Sung, Wing-Kin 414  
 Suri, Subhash 338
- Takahashi, Atsushi 365  
 Tang, Ganggui 128  
 Tarui, Jun 553  
 Thang, Nguyen Kim 247  
 Thankachan, Sharma V. 169  
 Tong, Weitian 353
- Uehara, Ryuhei 389, 553  
 Uno, Takeaki 94
- Valtr, Pavel 15  
 van't Hof, Pim 285  
 van Renssen, André 313  
 Verbeek, Kevin 338  
 Vu, Hoa 414
- Waleń, Tomasz 220  
 Wang, Haitao 3, 326  
 Wang, Hung-Lung 606  
 Wasa, Kunihiko 94  
 Watanabe, Osamu 273  
 Wiechert, Veit 516



- Witt, Carsten 686  
Wu, Yen-Wei 606  
Xiao, Mingyu 429  
Yamada, Takeshi 389  
Yang, Linji 41  
Yiu, Siu-Ming 414  
Yuan, Hao 41  
Zavershynskiy, Maksym 27  
Zeh, Norbert 128  
Zhang, Jia 661  
Zhang, Jialin 661  
Zhang, Jingru 3  
Zhou, Gelin 565  
Zhu, Daming 467, 491  
Zhu, Xuding 507