Evripidis Bampis   Klaus Jansen
Claire Kenyon (Eds.)

Efficient Approximation and Online Algorithms

Recent Progress on Classical Combinatorial Optimization Problems and New Applications

Springer

Volume Editors

Evripidis Bampis
Université d'Évry Val d'Essonne
LaMI, CNRS UMR 8042
523, Place des Terasses, Tour Evry 2, 91000 Evry Cedex, France
E-mail: bampis@lami.univ-evry.fr

Klaus Jansen
University of Kiel
Institute for Computer Science and Applied Mathematics
Olshausenstr. 40, 24098 Kiel, Germany
E-mail: kj@informatik.uni-kiel.de

Claire Kenyon
Brown University
Department of Computer Science
Box 1910, Providence, RI 02912, USA
E-mail: claire@cs.brown.edu

Library of Congress Control Number: 2006920093

CR Subject Classification (1998): F.2, C.2, G.2-3, I.3.5, G.1.6, E.5

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-32212-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-32212-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11671541 06/3142 5 4 3 2 1 0

Preface

In this book, we present some recent advances in the field of *combinatorial optimization* focusing on the design of *efficient approximation and on-line algorithms*. Combinatorial optimization and polynomial time approximation are very closely related: given an $\mathcal{NP}$-hard combinatorial optimization problem, i.e., a problem for which no polynomial time algorithm exists unless $\mathcal{P} = \mathcal{NP}$, one important approach used by computer scientists is to consider polynomial time algorithms that do not produce optimum solutions, but solutions that are provably close to the optimum. A natural partition of combinatorial optimization problems into two classes is then of both practical and theoretical interest: the problems that are *fully approximable*, i.e., those for which there is an approximation algorithm that can approach the optimum with any arbitrary precision in terms of relative error and the problems that are *partly approximable*, i.e., those for which it is possible to approach the optimum only until a fixed factor unless $\mathcal{P} = \mathcal{NP}$. For some of these problems, especially those that are motivated by practical applications, the input may not be completely known in advance, but revealed during time. In this case, known as the *on-line* case, the goal is to design algorithms that are able to produce solutions that are close to the *best* possible solution that can be produced by any *off-line* algorithm, i.e., an algorithm that knows the input in advance.

These issues have been treated in some recent texts [1], but in the last few years a huge amount of new results have been produced in the area of approximation and on-line algorithms. This book is devoted to the study of some classical problems of scheduling, of packing, and of graph theory, but also new optimization problems arising in various applications such as networks, data mining or classification. One central idea in the book is to use a linear program relaxation of the problem, randomization and rounding techniques.

The book is divided into 11 chapters. The chapters are self-contained and may be read in any order.

In Chap. 1, the goal is the introduction of a theoretical framework for dealing with data mining applications. Some of the most studied problems in this area as well as algorithmic tools are presented. Chap. 2 presents a survey concerning local search and approximation. Local search has been widely used in the core of many heuristic algorithms and produces excellent practical results for many combinatorial optimization problems. The objective here is to com-

[1] V. Vazirani, *Approximation Algorithms, Springer Verlag, Berlin, 2001*; G. Ausiello et al, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability, Springer Verlag, 1999*; D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, 1997*; A. Borodin, R. El-Yaniv, *On-line Computation and Competitive Analysis, Cambridge University Press, 1998*, A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art, LNCS 1442. Springer-Verlag, Berlin, 1998.*

pare from a theoretical point of view the quality of local optimum solutions with respect to a global optimum solution using the notion of the approximation factor and to review the most important results in this direction. Chap. 3 surveys the *wavelength routing problem* in the case where the underlying optical network is a tree. The goal is to establish the requested communication connections but using the smallest total number of wavelengths. In the case of trees this problem is reduced to the problem of finding a set of transmitter-receiver paths and assigning a wavelength to each path so that no two paths of the same wavelength share the same fiber link. Approximation and on-line algorithms, as well as hardness results and lower bound, are presented. In Chap. 4, a *call admission control problem* is considered in which the objective is the maximization of the number of accepted communication requests. This problem is formalized as an *edge-disjoint-path problem* in (non)-oriented graphs and the most important (non)-approximability results, for arbitrary graphs, as well as for some particular graph classes, are presented. Furthermore, combinatorial and linear programming algorithms are reviewed for a generalization of the problem, the *unsplittable flow problem.* Chap. 5 is focused on a special class of graphs, the intersection graphs of disks. Approximation and on-line algorithms are presented for the maximum independent set and coloring problems in this class. In Chap. 6, a general technique for solving min-max and max-min resource sharing problems is presented and it is applied to two applications: scheduling unrelated machines and strip packing. In Chap. 7, a simple analysis is proposed for the on-line problem of scheduling preemptively a set of tasks in a multiprocessor setting in order to minimize the flow time (total time of the tasks in the system). In Chap. 8, approximation results are presented for a general classification problem, the *labeling problem* which arises in several contexts and aims to classify related objects by assigning to each of them one label. In Chap. 9, a very efficient tool for designing approximation algorithms for scheduling problems is presented, the *list scheduling in order of α-points*, and it is illustrated for the single machine problem where the objective function is the sum of weighted completion times. Chap. 10 is devoted to the study of one classical optimization problem, the k-median problem from the approximation point of view. The main algorithmic approaches existing in the literature as well as the hardness results are presented. Chap. 11 focuses on a powerful tool for the analysis of randomized approximation algorithms, the Lovász-Local-Lemma which is illustrated in two applications: the job shop scheduling problem and resource-constrained scheduling.

We take the opportunity to thank all the authors and the reviewers for their important contribution to this book. We gratefully acknowledge the support from the EU Thematic Network APPOL I+II (Approximation and Online Algorithms). We also thank Ute Iaquinto and Parvaneh Karimi Massouleh from the University of Kiel for their help.

September 2005 Evripidis Bampis, Klaus Jansen, and Claire Kenyon

Table of Contents

On Approximation Algorithms for Data Mining Applications

Foto N. Afrati

National Technical University of Athens, Greece

Abstract. We aim to present current trends in the theoretical computer science research on topics which have applications in data mining. We briefly describe data mining tasks in various application contexts. We give an overview of some of the questions and algorithmic issues that are of concern when mining huge amounts of data that do not fit in main memory.

1 Introduction

Data mining is about extracting useful information from massive data such as finding frequently occurring patterns or finding similar regions or clustering the data. The advent of the internet has added new applications and challenges to this area. From the algorithmic point of view mining algorithms seek to compute good approximate solutions to the problem at hand. As a consequence of the huge size of the input, algorithms are usually restricted to making only a few passes over the data, and they have limitations on the random access memory they use and the time spent per data item.

The input in a data mining task can be viewed, in most cases, as a two dimensional $m \times n$ 0,1-matrix which often is sparse. This matrix may represent several objects such as a collection of documents (each row is a document and each column is a word and there is a 1 entry if the word appears in this document), or a collection of retail records (each row is a transaction record and each column represents an item, there is a 1 entry if the item was bought in this transaction), or both rows and columns are sites on the web and there is a 1 entry if there is a link from the one site to the other. In the latter case, the matrix is often viewed as a graph too. Sometimes the matrix can be viewed as a sequence of vectors (its rows) or even a sequence of vectors with integer values (not only 0,1).

The performance of a data mining algorithm is measured in terms of the number of passes, the required work space in main memory and computation time per data item. A constant number of passes is acceptable but one pass algorithms are mostly sought for. The workspace available ideally is constant but sublinear space algorithms are also considered. The quality of the output is usually measured using conventional approximation ratio measures [97], although in some problems the notion of approximation and the manner of evaluating the results remain to be further investigated.

These performance constraints call for designing novel techniques and novel computational paradigms. Since the amount of data far exceeds the amount of workspace available to the algorithm, it is not possible for the algorithm to “remember” large amounts of past data. A recent approach is to create a *summary* of the past data to store in main memory, leaving also enough memory for the processing of the future data. Using a random *sample* of the data is also another popular technique.

Besides data mining, other applications can be also modeled as one pass problems such as the interface between the storage manager and the application layer of a database system or processing data that are brought to desktop from networks, where each pass essentially is another expensive access to the network. Several communities have contributed (with technical tools and methods as well as by solving similar problems) to the evolving of the data mining field, including statistics, machine learning and databases.

Many single pass algorithms have been developed recently and also techniques and tools that facilitate them. We will review some of them here. In the first part of this chapter (next two sections), we review formalisms and technical tools used to find solutions to problems in this area. In the rest of the chapter we briefly discuss recent research in *association rules*, *clustering* and *web mining*. An association rule relates two columns of the entry matrix (e.g., if the i-th entry of a row v is 1 then most probably the j-th entry of v is also 1). Clustering the rows of the matrix according to various similarity criteria in a single pass is a new challenge which traditional clustering algorithms did not have. In web mining, one problem of interest in search engines is to rank the pages of the web according to their importance on a topic. Citation importance is taken by popular search engines according to which important pages are assumed to be those that are linked by other important pages.

In more detail the rest of the chapter is organized as follows. The next section contains formal techniques used for single pass algorithms and a formalism for the data stream model. Section 3 contains an algorithm with performance guarantees for finding approximately the L_p distance between two data streams. As an example, Section 4 contains a list of what are considered the main data mining tasks and another list with applications of these tasks. The last three sections discuss recent algorithms developed for finding association rules, clustering a set of data items and for searching the web for useful information. In these three sections, techniques mentioned in the beginning of the chapter are used (such as SVD, sampling) to solve the specific problems. Naturally some of the techniques are common, such as, for example, spectral methods are used in both clustering and web mining. As the area is rapidly evolving this chapter serves as a brief introduction to the most popular technical tools and applications.

2 Formal Techniques and Tools

In this section we present some theoretical results and formalisms that are often used in developing algorithms for data mining applications. In this context, the singular value decomposition (SVD) of a matrix (subsection 2.1) has inspired web search techniques, and, as a dimensionality reduction technique, is used for finding similarities among documents or clustering documents (known as the latent semantic indexing technique for document analysis). Random projections (subsection 2.1) offer another means for dimensionality reduction explored in recent work. Data streams (subsection 2.2) is proposed for modeling limited pass algorithms; in this subsection some discussion is done on lower and upper bounds on the required workspace. Sampling techniques (subsection 2.3) have also been used in statistics and learning theory, under somewhat different perspective however. Storing a sample of the data that fits in main memory and running a "conventional" algorithm on this sample is often used as the first stage of various data mining algorithms. We present a computational model for probabilistic sampling algorithms that compute approximate solutions. This model is based on the decision tree model [27] and relates the query complexity to the size of the sample.

We start by providing some (mostly) textbook definitions for self containment purposes. In data mining we are interested in vectors and their relationships under several distance measures. For two vectors, $\vec{v} = (v_1, \ldots, v_n)$, $\vec{u} = (u_1, \ldots, u_n)$, the *dot product* or *inner product* is defined to be a number which is equal to the sum of the component-wise products $\vec{v} \cdot \vec{u} = v_1u_1 + \ldots + v_nu_n$ and the L_p *distance* (or L_p *norm*) is defined to be: $||\vec{v} - \vec{u}||_p = (\Sigma_{i=1}^n |v_i - u_i|^p)^{1/p}$. For $p = \infty$, L_∞ distance is equal to $\max_{i=1}^n |u_i - v_i|$. The L_p distance is extended to be defined between matrices : $||\vec{V} - \vec{U}||_p = (\Sigma_i(\Sigma_j|V_{ij} - U_{ij}|^p))^{1/p}$. We sometimes use $||\ \ ||$ to denote $||\ \ ||_2$. The *cosine distance* is defined to be $1 - \frac{\vec{v} \cdot \vec{u}}{||\vec{v}||\ ||\vec{u}||}$. For sparse matrices the cosine distance is a suitable similarity measure as the dot product deals only with non-zero entries (which are the entries that contain the information) and then it is normalized over the lengths of the vectors.

Some results are based on *stable distributions* [85]. A distribution $\mathcal{D}$ over the reals is called *p-stable* if for any n real numbers $a_1, \ldots, a_n$ and independent identically distributed, with distribution $\mathcal{D}$, variables $X_1, \ldots, X_n$, the random variable $\Sigma_i a_i X_i$ has the same distribution as the variable $(\Sigma_i |a_i|^p)^{1/p} X$, where X is a random variable with the same distribution as the variables $X_1, \ldots, X_n$. It is known that stable distributions exist for any $p \in (0, 2]$. A Cauchy distribution defined by the density function $\frac{1}{\pi(1+x^2)}$, is 1-stable, a Gaussian (normal) distribution defined by the density function $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable.

A *randomized algorithm* [81] is an algorithm that flips coins, i.e., it uses random bits, while no probabilistic assumption is made on the distribution of the input. A randomized algorithm is called *Las-Vegas* if it gives the correct answer on all inputs. Its running time or workspace could be a random variable depending on the random variable of the coin tosses. A randomized algorithm is called *Monte-Carlo with error probability* ϵ if on every input it gives the right answer with probability at least $1 - \epsilon$.

2.1 Dimensionality Reduction

Given a set S of points in the multidimensional space, dimensionality reduction techniques are used to map S to a set S' of points in a space of much smaller dimensionality while approximately preserving important properties of the points in S. Usually we want to preserve distances. Dimensionality reduction techniques can be useful in many problems where distance computations and comparisons are needed. In high dimensions distance computations are very slow and moreover it is known that, in this case, the distance between almost all pairs of points is the same with high probability and almost all pair of points are orthogonal (known as the Curse of Dimensionality).

Dimensionality reduction techniques that are popular recently include Random Projections and Singular Value Decomposition (SVD). Other dimensionality reduction techniques use linear transformations such as the Discrete Cosine transform or Haar Wavelet coefficients or the Discrete Fourier Transform (DFT). DFT is a heuristic which is based on the observation that, for many sequences, most of the energy of the signal is concentrated in the first few components of DFT. The L_2 distance is preserved exactly under the DFT and its implementation is also practically efficient due to an $O(nlogn)$ DFT algorithm.

Dimensionality reduction techniques are well explored in databases [51,43].

Random Projections. Random Projection techniques are based on the Johnson-Lindenstrauss (JL) lemma [67] which states that any set of n points can be embedded into the k-dimensional space with $k = O(\log n/\epsilon^2)$ so that the distances are preserved within a factor of ϵ.

Lemma 1. *(JL) Let $\vec{v}_1, \ldots, \vec{v}_m$ be a sequence of points in the d-dimensional space over the reals and let $\epsilon, F \in (0, 1]$. Then there exists a linear mapping f from the points of the d-dimensional space into the points of the k-dimensional space where $k = O(\log(1/F)/\epsilon^2)$ such that the number of vectors which approximately preserve their length is at least $(1 - F)m$. We say that a vector $\vec{v}_i$ approximately preserves its length if:*

$$||\vec{v}_i||^2 \leq ||f(\vec{v}_i)||^2 \leq (1 + \epsilon)||\vec{v}_i||^2$$

The proof of the lemma, however, is non-constructive: it shows that a random mapping induces small distortions with high probability. Several versions of the proof exist in the literature. We sketch the proof from [65]. Since the mapping is linear, we can assume without loss of generality that the $\vec{v}_i$'s are unit vectors. The linear mapping f is given by a $k \times d$ matrix $\vec{A}$ and $f(\vec{v}_i) = \vec{A}\vec{v}_i, i = 1, \ldots, m$. By choosing the matrix $\vec{A}$ at random such that each of its coordinates is chosen independently from $N(0, 1)$, then each coordinate of $f(\vec{v}_i)$ is also distributed according to $N(0, 1)$ (this is a consequence of the spherical symmetry of the normal distribution). Therefore, for any vector $\vec{v}$, for each $j = 1, \ldots, k/2$, the sum of squares of consecutive coordinates $Y_j = ||f(\vec{v})_{2j-1}||^2 + ||f(\vec{v})_{2j}||^2$ has exponential distribution with exponent $1/2$. The expectation of $L = ||f(\vec{v})||^2$ is equal to $\Sigma_j E[Y_j] = k$. It can be shown that the value of L lies within ϵ of its mean with probability $1 - F$. Thus the expected number of vectors whose length is approximately preserved is $(1 - F)m$.

The JL lemma has been proven useful in improving substantially many approximation algorithms (e.g., [65,17]). Recently in [40], a deterministic algorithm is presented which finds such mapping in time almost linear in the number of distances to preserve times the dimension d of the original space.

In recent work, random projections are used to compute summaries of past data called *sketches* to solve problems such as approximating the L_p norm of a data stream (see also section 3).

Singular Value Decomposition. Consider matrices with real numbers as entries. We say that a matrix $\vec{M}$ is *orthogonal* if $\vec{M}\vec{M}^{Tr} = \vec{I}$ where $\vec{I}$ is the identity matrix (by $\vec{A}^{Tr}$ we denote the transpose of matrix $\vec{A}$). An *eigenvalue* of a $n \times n$ matrix $\vec{M}$ is a number λ such that there is a vector $\vec{t}$ which satisfies $\vec{M}\vec{t} = \lambda\vec{t}$. Such a vector $\vec{t}$ is called an *eigenvector* associated with λ. The set of all eigenvectors associated with λ form a subspace and the dimension of this subspace is called the *multiplicity* of λ. If $\vec{M}$ is a symmetric matrix, then the multiplicities of all eigenvalues sum up to n. Let us denote all the eigenvalues of such a matrix $\vec{M}$ by $\lambda_1(\vec{M}), \lambda_2(\vec{M}), \ldots \lambda_n(\vec{M})$, where we have listed each eigenvalue a number of times equal to its multiplicity. For symmetric matrix $\vec{M}$, we can choose for each $\lambda_i(\vec{M})$ an associated eigenvector $\vec{t_i}(\vec{M})$ such that the set of vectors $\{\vec{t_i}(\vec{M})\}$ forms an orthonormal basis for the n-dimensional space over the real numbers. Let $\vec{Q}$ be the matrix with columns these vectors and let Λ be the diagonal matrix with diagonal entries the list of eigenvalues. Then, it is easy to prove that: $\vec{M} = \vec{Q}\Lambda\vec{Q}^{Tr}$. However the result extends to any matrix as the following theorem states.

Theorem 1. *(Singular Value Decomposition/SVD) Every $m \times n$ matrix $\vec{A}$ can be written as $\vec{A} = \vec{U}\vec{T}\vec{V}^{Tr}$ where $\vec{U}$ and $\vec{V}$ are orthogonal and $\vec{T}$ is diagonal.*

The diagonal entries of $\vec{T}$ are called the *singular values* of $\vec{A}$. It is easy to verify that the columns of $\vec{U}$ and $\vec{V}$ represent the eigenvectors of $\vec{A}\vec{A}^{Tr}$ and $\vec{A}^{Tr}\vec{A}$ respectively and the diagonal entries of $\vec{T}^2$ represent their common set of eigenvalues. The importance of the SVD in dimensionality reduction lies in the following theorem which states that $\vec{U}$, $\vec{T}$, $\vec{V}$ can be used to compute, for any k, the matrix A_k of rank k which is "closest" to $\vec{A}$ over all matrices of rank k.

Theorem 2. *Let the SVD of $\vec{A}$ be given by $\vec{A} = \vec{U}\vec{T}\vec{V}^{Tr}$. Suppose $\tau_1, \ldots, \tau_k$ are the k largest singular values. Let $\vec{u}_i$ be the i-th column of $\vec{U}$ and $\vec{v}_i$ be the i-th column of $\vec{V}$ and let τ_i be the i-th element in the diagonal of $\vec{T}$. Let r be the rank of $\vec{A}$ and let $k < r$. If*

$$\vec{A}_k = \Sigma_{i=1}^{k} \tau_i \vec{u}_i \vec{v}_i^{Tr}$$

Then

$$\min_{rank(\vec{B})=k} ||\vec{A} - \vec{B}||_2 = ||\vec{A} - \vec{A}_k||_2 = \tau_{k+1}$$

The SVD technique displays optimal dimensionality reduction (for linear projections) but it is hard to compute.

2.2 The Data Stream Computation Model

The streaming model is developed to formalize a single (or few) pass(es) algorithm over massive data that do not fit in main memory. In this model, the data is observed once (or few times) and in the same order it is generated. For each data item, we want to minimize the required workspace and the time to process it.

In the interesting work of [61] where the stream model was formalized, a *data stream* is defined as a sequence of data items $v_1, v_2, \ldots, v_n$ which are assumed to be read by an algorithm only once (or very few times) in increasing order of the indices i. The number P of *passes* over the data stream and the *workspace* W (in bits) required by the algorithm in the main memory are measured. The performance of an algorithm is measured by the number of passes the algorithm makes over the data and the required workspace, along with other measures such as the computation time per input data item. This model does not necessarily require a bound on the computation time.

Tools from communication complexity are used to show lower bounds on the workspace of limited-pass algorithms [8],[61]. *Communication complexity* [79] is defined as follows. In the *(2-party) communication model* there are two players A and B. Player A is given a x from a finite set X and player B is given a y from a finite set Y. They want to compute a function $f(x, y)$. As player A does not know y and player B does not know x, they need to communicate. They use a protocol to exchange bits. The *communication complexity of a function* f is the minimum over all communication protocols of the maximum over all $x \in X, y \in Y$ of the number of bits that need to be exchanged to compute $f(x, y)$. The protocol can be deterministic, Las-Vegas or Monte-Carlo. If one player is only transmitting and one is only receiving then it is called *one-way communication complexity.* In this case, only the receiver needs to be able to compute function f.

To see how communication complexity is related to deriving lower bounds on the space, think of one way communication where player A has the information of the past data and player B has the information of the future data. The communication complexity can be used as a lower bound on the space available to store a "summary" of the past data.

It is natural to ask whether under the stream model there are noticeable differences regarding the workspace requirements (i) between one-pass and multi-pass algorithms, (ii) between deterministic and randomized algorithms and (iii) between exact and approximation algorithms. These questions were explored in earlier work [82] in context similar to data streams and it was shown that: (i) Some problems require a large space in one pass and a small space in two passes. (ii) There can be an exponential gap in space bounds between Monte-Carlo and Las-Vegas algorithms. (iii) For some problems, an algorithm for an approximate solution, requires substantially less space than an exact solution algorithm.

In [8], space complexity for estimating the frequency moments of a sequence of elements in one pass was studied and tight lower bounds were derived. The problem studied in [82] is the space required for selecting the k-th largest out of

n elements using at most P passes over the data. An upper bound of $n^{1/P} \log n$ and a lower bound of $n^{1/P}$ is shown, for large enough k. Recent work on space lower bounds includes also [90].

The data stream model appears to be related to other work e.g., on competitive analysis [69], or I/O efficient algorithms [98]. However, it is more restricted in that it requires that a data item can never again be retrieved in main memory after its first pass (if it is a one-pass algorithm). A distributed stream model is also proposed in [53] which combines features of both streaming models and communication complexity models.

Streaming models have been extensively studied recently and methods have been developed for comparing data streams under various L_p distances, or clustering them. The stream model from the database perspective is investigated in the Stanford stream data management Project [93] (see [11] for an overview and algorithmic considerations).

2.3 Sampling

Randomly sampling a few data items of a large data input is often a technique used to extract useful information about the data. A small sample of the data may be sufficient to compute many statistical parameters of the data with reasonable accuracy. Tail inequalities from probability theory and the central limit theorem are useful here [81,47].

One of the basic problems in this context is to computes the size of the sample required to determine certain statistical parameters. In many settings, the size of the sample for estimating the number of distinct values in a data set is of interest. The following proposition [86] gives a lower bound on the size of the sample in such a case whenever we know the number of distinct values and each has a frequency greater than ϵ.

Proposition 1. *If a dataset D contains $l \geq k$ distinct values of frequency at least ϵ, then a sample of size $s \geq \frac{1}{\epsilon} \log \frac{k}{\delta}$ contains at least k distinct values with probability $> 1 - \delta$.*

To prove, let $a_1, \ldots, a_l$ be the l distinct values of frequencies $p_1, \ldots, p_l$ respectively and, each frequency is at least ϵ. Then the probability our sample missed k of these distinct values is at most $\Sigma_{i=1}^{k} (1 - p_i)^s \leq k(1 - \epsilon)^s \leq \delta$ by our choice of s.

In a similar context, random sampling from a dataset whose size is unknown, is of interest in many applications. The problem is to select a random sample of size n from a dataset of size N when N is unknown. A one-pass *reservoir* algorithm is developed in [99]. A *reservoir* algorithm maintains a sample (reservoir) of data items in main memory and data items may be selected for the reservoir as they are processed. The final random sample will be selected from the sample maintained in the reservoir (hence the size of the sample in the reservoir is larger than n). In [99] each data item is selected with probability M/n where n is the number of data items read so far and M is the size of the reservoir.

An algorithm that uses a sample of the input is formalized in [14] as a *uniform randomized decision tree*. This formalism is used to derive lower bounds on the required size of the sample. A *randomized decision tree* has two kinds of internal nodes, *query* nodes and *random coin toss* nodes. Leaves are related to output values. On an input $x_1, \dots, x_n$, the computation of the output is done by following a path from the root to a leaf. On each internal node a decision is made as to which of its children the computation path moves next. In a random coin toss node this decision is based on a coin toss which picks one of the children uniformly at random. A query node v has two labels, an input location (to be queried), and a function which maps a sequence of query locations (the sequence is thought of as the input values queried so far along the path from the root) to one of the children of this node v. The child to which the path moves next is specified by the value of this function. Each leaf is labeled by a function which maps the sequence of query locations read along the path to an output value. The output is the value given by the function on the leaf which is the end point of the computation path. Note that any input $x_1, \dots, x_n$ may be associated with several possible paths leading from the root to a leaf, depending on the random choices made in the random coin nodes. These random choices induce a distribution over the paths corresponding to $x_1, \dots, x_n$.

A *uniform* randomized decision tree is defined as a randomized decision tree with the difference that each query node is not labeled by an input variable. The query in this case is done uniformly at random over the set of input values that have not been queried so far along the path from the root. A uniform decision tree can be thought as a sampling algorithm which samples the input uniformally at random and uses only these sample values to decide the output. Thus the number of query nodes along a path from the root to a leaf is related to the size of the sample.

The *expected query complexity of a decision tree T on input $\vec{x} = x_1, \dots, x_n$* denoted $S^e(T, \vec{x})$, is the expected number of query nodes on paths corresponding to $\vec{x}$. The *worst case query complexity of a tree T on input $\vec{x}$*, denoted $S^w(T, \vec{x})$, is the maximum number of query nodes on paths corresponding to $\vec{x}$. Here the expectation and the maximum are taken over the distribution of paths. The *expected and worst case query complexity of T* $S^e(T)$ and $S^w(T)$ are the maximum of $S^e(T, \vec{x})$ and $S^w(T, \vec{x})$, respectively, over all inputs $\vec{x}$ in A^n.

Because of the relation between query complexity and the size of the required sample, a relationship can also be obtained between query complexity and space complexity as defined in the data stream model. Let $\epsilon \geq 0$ be an error parameter, δ $(0 < \delta < 1)$ a confidence parameter, and f a function. A decision tree is said to (ϵ, δ)-*approximate* f if for every input $\vec{x}$ the probability of paths corresponding to $\vec{x}$ that output a values y within a factor of ϵ from the exact solution is at least $1 - \delta$. The (ϵ, δ) *expected query complexity* of f is:

$$S^e_{\epsilon,\delta}(f) = \quad min\{S^e(T) \mid T \quad (\epsilon, \delta) - approximates \quad f\}$$

The *worst case query complexity of a function f* is defined similarly.

The (ϵ, δ) query complexity of a function f can be directly related to the space complexity as defined on data streams. If a function has (ϵ, δ) query complexity

$S^e_{\epsilon,\delta}(f)$, then the space required in main memory is at most $S^w_{\epsilon,\delta}(f)O(\log|A| + \log n)$, where A, n are parameters of the input vector $\vec{x}$. For input vector $\vec{x} = x_1, \ldots, x_n$, n is the number of data items and A is the number of elements from which the values of each x_i is drawn.

Based on this formalization, a lower bound is obtained on the number of samples required to distinguish between two distributions [14]. It it also shown that the k-th statistical moment can be approximated within an additive error of ϵ by using a random sample of size $O(1/\epsilon^2 \log \frac{1}{\delta})$, and that this is a lower bound on the size of the sample.

Work that also refer to lower bounds on query complexity for approximate solutions include results on the approximation of the mean [28], [36,91], the approximation on the frequency moment [31].

Lossy compression may be related to sampling. When we have files in compressed form, we might want to compute functions of the uncompressed file without having to decompress. Compressed files might even been thought of as not been able to be precisely retrieved by decompression, namely the compression (in order to gain larger compression factors) allowed for some loss of information (*lossy compression*). Problems of this nature are related to sampling algorithms in [46].

Statistical decision theory and statistical learning theory are fields where sampling methods are used too. However they focus on different issues than data mining does. Statistical decision theory [16] studies the process of making decisions based on information gained by computing various parameters of a sample. However the sample is assumed given and methods are developed that maximize the utitily of it. Computing the required size of a sample for approximately computing parameters of the input data is not one of its concerns. Statistical learning theory [96,70] is concerned with learning an unknown function from a class of target functions, i.e., approximating the function rather, whereas, in data mining, the interest is in approximating some parameter of the function.

For an excellent overview on key research results and formal techniques on data stream algorithms see the tutorial in [51] and references therein. Also an excellent survey on low distortion embedding techniques for dimensionality reduction can be found in [63].

3 Approximating the L_p Distance. Sketches

We consider in this section the following problem which may be part of various data mining tasks. The data stream model is assumed and we want to compute an approximation to the L_p distance. Formally, we are given a stream S of data items. Each data item is viewed as a pair (i, v), $i = 1, \ldots, n$, with entries for v an integer in the range $\{-M, M\}$ where M is a positive integer (so we need $\log M$ memory to store the value of each data item). Note that there may exist several pairs (with possibly different values for v) for a specific i. We want to compute a good approximation of the following quantity:

$$L_p(S) = (\Sigma_{i=1,\ldots,n}|\Sigma_{(i,v)\in S} v|^p)^{1/p}$$

The obvious solution to this problem, i.e., maintain a counter for each i is too costly because of the size of the data. In the influential paper [8], a scheme is proposed for approximating $L_2(S)$ within a factor of ϵ in workspace $O(1/\epsilon)$ with arbitrarily large constant probability.

In [46], a solution for $L_1(S)$ is investigated for the special case where there are at most two non zero entries for each i. In this case, the problem can be equivalently viewed as having two streams S_a and S_b and asking for a good approximation of $L_1(S_a, S_b) = \Sigma_i |\Sigma_{(i,v)\in S_a} v - \Sigma_{(i,v) v\in S_b} v|$. A single pass algorithm is developed which, with probability $1-\delta$, computes an approximation to $L_1(S)$ within a factor of ϵ using $O(\log M \log n \log(1/\delta)/\epsilon^2)$ random access space and $O(\log n \log\log n + \log M \log(1/\delta)/\epsilon^2)$ computation time per item. The method in [46] can be viewed as using *sketches of vectors*, which is a *summary data structure*. In this case, a sketch $C(S_a)$, $C(S_b)$ is computed for each data stream S_a, S_b respectively. Sketches are much smaller in size than S_a, S_b and such that an easily computable function of the sketches gives a good approximation to $L_1(S_a, S_b)$.

In [62], a unifying framework is proposed for approximating $L_1(S)$ and $L_2(S)$ within a factor of ϵ (with probability $1-\delta$) using $O(\log M \log(n/\delta)\log(1/\delta)/\epsilon^2)$ random access space and $O(\log(n/\delta))$ computation time per item. The technique used combines the use of stable distributions [85] with Nisan pseudorandom generators [84]. The property of stable distributions which is used in this algorithm is the following. The dot product of a vector $\vec{u}$ with a sequence of n independent identically distributed random variables having p-stable distribution is a good estimator of the L_p norm of $\vec{u}$. In particular we can use several such products to embed a d-dimensional space into some other space (of lower dimensionality) such that to approximately preserve the L_p distances. Dot product can be computed in small workspace.

We shall describe here in some detail the first stage of this algorithm for approximating $L_1(S)$: For $l = O(c/\epsilon^2 \log 1/\delta)$ (for some suitable constant c), we initialize nl independent random variables $X_i^j, i = 1, \ldots, n, j = 1, \ldots, l$ with Cauchy distribution defined by the density function $f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$ (we know this distribution is 1-stable). Then, the following three steps are executed:

1. Set $S^j = 0$, for $j = 1, \ldots, l$.
2. For each new pair (i, v) do: $S^j = S^j + vX_i^j$ for all $j = 1, \ldots, l$.
3. Return the $median(|S^0|, \ldots, |S^{l-1}|)$.

To prove the correctness of this algorithm we argue as follows: We want to compute $L_1(S) = C = \Sigma_i |c_i|$ where $c_i = \Sigma_{(i,v)\in S} v$. First, it follows from the 1-stability of the Cauchy distribution that, each S^j has the same distribution as CX where X has Cauchy distribution. Random variable X has Cauchy distribution with density function $f(x) = \frac{1}{\pi}\frac{1}{1+x^2}$ hence $median(|X|) = 1$ and $median(v|X|) = v$ for any v. It is known that for any distribution, if we take $l = O(c/\epsilon^2 \log 1/\delta)$ independent samples and compute the median M, then for distribution function $F(M)$ of M we have (for a suitable constant c) $Pr[F(X) \in$

$[1/2 - \epsilon, 1/2 + \epsilon]] > 1 - \delta$. Thus, it can be proven that $l = O(c/\epsilon^2 \log 1/\delta)$ independent samples approximate $L_1(S)$ within a factor of ϵ with probability $> 1 - \delta$.

This stage of the algorithm, though, assumes random numbers of *exact* precision. Thus, random generators are used to solve the problem of how to reduce the number of required random bits.

The problem of approximating L_p distances in one pass algorithms has a variety of applications including estimation of the size of self join [8,52] and estimation of statistics of network flow data [46].

In the above frameworks a solution was facilitated by using summary descriptions of the data which approximately preserved the L_p distances. The summaries were based on computing with random variables. Techniques that use such summaries to reduce the size of the input are known as *sketching* techniques (they compute a sketch of each input vector). Computing sketches has been used with success in many problems to get summaries of the data. It has enabled compression of data and has been speeding up computation for various data mining tasks [64,34,25,26,35]. (See also section 5 for a description of the algorithm in [34].) Sketches based on random projections are often used to approximate L_p distances or other measure of similarities depending on them. In such a case (see e.g., [35]) sketches are defined as: The i-th component of the sketch $\vec{s}(\vec{x})$ of $\vec{x}$ is the dot product of $\vec{x}$ with a random vector $\vec{r_i}$: $\vec{s_i}(\vec{x}) = \vec{x} \cdot \vec{r_i}$, where each component of each random vector is drawn from a Cauchy distribution. Work that use sketching techniques include [38,49] where aggregate queries and multi-queries over data streams are computed.

4 Data Mining Tasks and Applications

The main data mining tasks are considered to be those that have an almost well defined algorithmic objective and assume that the given data are cleaned. In this section, we mention some areas of research and applications that are considered of interest in the data mining community [59]. We begin with a list of the most common data mining tasks:

– *Association rules:* Find correlations among the columns of the input matrix of the form: if there is a 1 entry in column 5 then most probably there is a 1 entry in column 7 too. These rules are probabilistic in nature.
– *Sequential patterns:* Find sequential patterns that occur often in a dataset.
– *Time series similarity:* Find criteria that check in a useful way whether two sequences of data exhibit "similar features".
– *Sequence matching:* Given a collection of sequences and a sequence query, find the sequence which is closest to the query-sequence.
– *Clustering:* Partition a given set of points into groups, called clusters so that "similar" points belong to the same cluster. A measure of similarity is needed, often it is a distance in a metric space.
– *Classification:* Given a set of points and a set of labels, assign labels to point so that similar objects are labeled by similar labels and a point is labeled

by the most likely label. A measure of similarity of points and similarity of labels is assumed and a likelihood of a point to be assigned a particular label.

– *Discovery of outliers:* Discover points in the dataset which are isolated, i.e., they do not belong to any multi-populated cluster.
– *Frequent episodes:* An extension of sequential pattern finding, where more complex patterns are considered.

These tasks as described in the list are usually decomposed in more primitive modules that may be common in several tasks, e.g., comparing large pieces of the input matrix to find similarity patterns is useful in clustering, and association rules mining.

We also include a list of some of the most common applications of data mining:

– *Marketing.* Considered one of the most known successes of data mining. Market basket analysis is motivated by the decision support problem and aims at observing customer habits to decide on business policy regarding prices or product offers. Basket data are collected by most large retail organizations and used mostly for marketing purposes. In this context, it is of interest to discover association rules such as ”if a person buys pencil then most probably buys paper too”. Such information can be used to increase sales on pencils by placing them near paper or make a profit by offering good prices on pencils and increase the price of paper.
– *Astronomy.* Clustering celestial objects by their radiation to distinguish galaxies and other star formations.
– *Biology.* Correlate diabetes to the presence of certain genes. Find DNA sequences representing genomes (sequential patterns). Work in time series analysis has many practical applications here.
– *Document analysis.* Cluster documents by subject. Used, for example, in collaborative filtering, namely tracking user behavior and making recommendations to individuals based on similarity of their preferences to these of other users.
– *Financial Applications.* Use time series similarity to find stocks with similar price movements or find products with similar selling patterns. Observe similar patterns in customers' financial history to decide if a bank loan is awarded.
– *Web mining.* Search engines like Google rank web pages by their “importance” in order to decide the order on which to present search results on a user query. Identifying communities on the web, i.e., groups that share a common interest and have a large intersection of web pages that are most often visited by the members of a group is another interesting line of research. This may be useful for advertising or to identify the most up-to-date information on a topic or to provide a measure of page rank which is not easy to spam. One popular method is to study co-citation and linkage statistics: web communities are characterized by dense directed bipartite subgraphs.
– *Communications.* Discover the geographic distribution of cell phone traffic at different base stations or the evolution of traffic at Internet routers over

time. Detecting similarity patterns over such data is important, e.g., which geographic regions have similar cell phone usage distribution, or which IP subnet traffic distributions over time intervals are similar.
– *Detecting intrusions.* Detecting intrusions the moment they happen is important to protecting a network from attack. Clustering is a technique used to detecting intrusions.
– *Detecting failures in network.* Mining episodes helps to detect faults in electricity network before they occur or detect congestions in packet switched networks.

The data available for mining interesting knowledge (e.g., census data, corporate data, biological data) is often in bad shape (have been gathered under no particular considerations), e.g. it may contain duplicate or incomprehensible information. Therefore a preprocessing stage is required to clean the data. Moreover after the results of a data mining task are obtained they may need a post processing stage to interprete and visualize them.

5 Association Rules

Identifying association rules in market basket data is considered to be one of the most well known successes of the data mining field. The problem of mining for association rules and the related problem of finding frequent itemsets have been studied extensively and many efficient heuristics are known. We will mention some of them in this section.

Basket data is a collection of records (or baskets), each record typically consisting of a transaction date and a collection of items (thought of as the items bought in this transaction). Formally we consider a domain set $\mathcal{I} = \{i_1, \ldots, i_m\}$ of elements called *items* and we are given a set $\mathcal{D}$ of transactions where each transaction T is a subset of $\mathcal{I}$. We say that a transaction T contains a set X of items if $X \subseteq T$. Each transaction is usually viewed as a row in a $n \times k$ 0,1-matrix where 1 means that the item represented by this column is included in this transaction and 0 that it is not included. The rows represent the baskets and the columns represent the items in the domain. The columns are sometimes called attributes or literals. Thus an instance of market basket data is represented by a 0,1-matrix.

The problem of mining association rules over basket data was introduced in [4]. An *association rule* is an "implication" rule $X \Rightarrow Y$ where $X \subset \mathcal{I}$ and $Y \subset \mathcal{I}$ and X, Y are disjoint. The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* c if $c\%$ of the transactions in $\mathcal{D}$ that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set $\mathcal{D}$ if $s\%$ of the transactions in $\mathcal{D}$ contain $Y \cup X$. The symbol $\Rightarrow$ used in an association rule is not a logical implication, it only denotes that the confidence and the support are estimated above the thresholds $c\%$ and $s\%$ respectively. In this context, the problem of mining for association rules on a given transaction set asks to generate all association rules with confidence and support thresholds greater than two given integers.

Functional dependencies are association rules with confidence 100% and any support and they are denoted as $X \rightarrow A$. Consequently, having determined a dependency $X \rightarrow A$, any dependency of the form $X \cup Y \rightarrow A$ can be ignored as redundant. The general case of association rules however is probabilistic in nature. Hence a rule $X \Rightarrow A$ does not make rule $X \cup Y \Rightarrow A$ redundant because the latter may not have minimum support. Similarly, rules $X \Rightarrow A$ and $A \Rightarrow Z$ do not make rule $X \Rightarrow Z$ redundant because the latter may not have minimum confidence.

In the context of the association rule problem, mining for frequent itemsets is one of the major algorithmic challenges. The *frequent itemsets problem* asks to find all sets of items (*itemsets*) that have support above a given threshold. This problem can be reduced to finding all the *maximal frequent itemsets* due to the monotonicity property – i.e., any subset of a frequent itemset is a frequent itemset too. A frequent itemset is maximal if any itemset which contains it is not frequent.

5.1 Mining for Frequent Itemsets

The monotonicity property has inspired a large number of algorithms known as a-priori algorithms which use the following a-priori trick: The algorithms begin the search for frequent itemsets with searching for frequent items and then construct candidate pairs of items only if both items in the pair are frequent. In the same fashion, they construct frequent candidate triples of items only if all the three pairs of items in the triple are found frequent in the previous step. Thus, to find frequent itemsets, they proceed levelwise, finding first the frequent items (sets of size 1), then the frequent pairs, the frequent triples, and so on.

An *a-priori algorithm* [4,6] needs to store the frequent itemsets found in each level in main memory (it assumes that there is enough space) so that to create the candidate sets for next level. It needs so many passes through the data as the maximum size of a frequent itemset or two passes if we are only interested in frequent pairs as is the case in some applications. Improvements have been introduced in this original idea which address issues such as: if the main memory is not enough to accommodate counters for all pairs of items, then e.g., hashing is used to prune some infrequent pairs in the first pass.

In [21], the number of passes is reduced by taking a dynamic approach to the apriori algorithm which is called Dynamic Itemset Counting. It reduces the number of passes of apriori by starting counting 2-itemsets (and possibly 3-itemsets) during the first pass. After having read (say) one third of the data, it builds candidate 2-itemsets based on the frequent 1-itemsets count so far. Thus running on the rest two thirds of the data, it checks also the counts of these candidates and it stops checking the 2-itemsets counts during the second pass after having read the first third of data. Similarly, it may start considering 3-itemsets during the first pass after having read the first two thirds of the data and stops considering them during the second run. If the data is fairly homogeneous, this algorithm finds all frequent itemsets in around two passes.

In [89], a hash table is used to determine on the first pass (while the frequent items are being determined) that many pairs are not possibly frequent (assuming that there is enough main memory). The hash table is constructed so that each of its buckets stores the accumulative counts of more than one pairs. This algorithm works well when infrequent pairs have small counts so that even when all the counts of pairs in the same bucket are added, the result is still less than the threshold. In [44], multiple hash tables are used in the first pass and a candidate pair is required to be in a large bucket in *every* hash table. In the second pass another hash table is used to hash pairs and in the third pass, only if a pair belongs to a frequent bucket in pass two (and has passed the test of pass one too) is taken as a candidate pair. The multiple hash tables improve the algorithm when most of the buckets have counts a lot below the threshold (hence many buckets are likely to be small).

These methods, however, cannot be used for finding all frequent itemsets in one or two passes. Algorithms that find all frequent itemsets in one or two passes usually rely on randomness of data and sampling. A simple approach is to take a main-memory-sized sample of the data, run one of the main-memory algorithms, find the frequent itemsets and either stop or run a pass through the data to verify. Some frequent itemsets might be missed in this way. In [94] this simple approach is taken. The algorithm on main memory is run on a much lower threshold so it is unlikely that it will miss a frequent itemset. To verify, we add to the candidates of the sample the negative border: an itemset S is in the negative border if S is not identified as frequent in the sample, but every immediate subset of S is. The candidate itemsets includes all itemsets in the negative border. Thus the final pass through the data counts the frequency of the itemsets in the negative border. If no itemset in the negative border is frequent, then the sample has given all the frequent itemsets candidates. Otherwise, we may rerun the whole procedure if we do not want to miss any frequent itemset.

A large collection of algorithms have been developed for mining itemsets in various settings. Recent work in [71] provides a unifying approach for mining constrained itemsets, i.e., under a more general class of constraints than the minimuum support constraint. The approach is essentially a generalization of the a priori principle.

Another consideration in this setting is that the collection of frequent itemsets found may be large and hard to visualize. Work done in [2] shows how to approximate the collection by a simpler bound without introducing many false positives and false negatives.

5.2 Other Measures for Association Rules

However confidence and support are not the only measures of "interestingness" of an association rule and do not always capture the intuition. Confidence is measured as the conditional probability of X given Y and it ignores the comparison to the (unconditional) probability of X. If the probability of X is high then confidence might be measured above threshold although this would not imply any correlation among X and Y. Other measures considered are the *interest* and the *conviction* [21]. Interest is defined as the probability of both X and Y divided by the product of the probability of X times the probability of Y. It is symmetric with respect to X and Y, and measures their correlation (or how far they are from being statistically independent). However, it can not derive an implication rule (which is non-symmetric). Conviction is defined as a measure closer to the intuition of an implication rule $X \Rightarrow Y$. Its motivation comes from the observation that if $X \Rightarrow Y$ is viewed as a logical implication, then it can be equivalently written as $\neg(X \wedge \neg Y)$. Thus conviction measures how far from statistical independence are the facts X and $\neg Y$ and is defined as follows: $\frac{P(X)P(\neg Y)}{P(X,\neg Y)}$.

In [20] conditional probability is not used to measure interestingness of an association rule and propose statistical correlation instead. In [92], *causal* rules instead of mere associations are discussed aiming to capture the intuition whether $X \Rightarrow Y$ means that X causes Y or some other item causes them both to happen. This direction of investigation is taken by noticing that yielding a small number (possibly arbitrarily decided) of the "most interesting" causal relationships might be desirable in many data mining contexts, since exploratory analysis of a dataset is what is usually the aim of a data mining task. In that perspective, it is pointed out that ongoing research in Bayesian learning (where several techniques are developed to extract causal relationships) seems promising for large scale data mining.

5.3 Mining for Similarity Rules

As pointed out, various other kinds of rules may be of interest given a set of basket data. A *similarity rule* $X \simeq Y$ denotes that the itemsets X and Y are highly correlated, namely they are contained both in a large fraction of the transactions that contain either X or Y. A similarity rule does not need to satisfy a threshold on the support, low-support rules are also of interest in this setting. Although for market basket analysis, the low support mining might not be very interesting, when the matrix represents the web graph, then similar web sites with low support might encompass similar subjects or mirror pages or plagiarism (in this case, rows will be sentences and columns web pages).

As low support rules are also of interest, techniques with support pruning (like finding all frequent itemsets) are not of use. However, in cases where the number of columns is sufficiently small then we can store something per column in main memory. A family of algorithms were developed in [34] to solve the problem in those cases using a hashing techniques.

For each column C, a signature $S(C)$ is defined which, intuitively, is a summary of the column. Signatures are such that a) they are small enough such that a signature for each column can fit in main memory and, b) similar columns have similar signatures. When the matrix is sparse, we cannot choose a small number of rows at random and use each shortened column in this set of rows as the signature. Most likely almost all signatures will be all 0's. The idea in this paper is: For each pair of columns, ignore the rows that both columns have zero entries, find the fraction of rows that these columns differ (over all non-both-zero-entry rows) and define this as the similarity measure. Interestingly, it can be proven that this similarity measure is proposrtional to the probability that both rows have the first occurrence of 1 in the same row. Thus the signature of each column is defined as the index of the first row with a 1 entry. Based on this similarity measure, two techniques that are developed in [34] are Min-Hashing (inspired by an idea in [33] –see also [24]) and Locality-Sensitive Hashing (inspired by ideas used in [56]–see also [65]).

In Min-Hashing, columns are hashed to the same bucket if they agree on the index of the first row with a 1 entry. To reduce the probability of false positives and false negatives, a set of p signatures are collected instead of one signature. This is done by implicitly considering a set of p different random permutations of the rows and for each permutation get a signature for each column. For each column, we use as its new signature the sequence of the p row indices (the row where the first 1 entry appears in this column). Actually these p row indices can be derived using only one pass through the data by hashing each row using p different hash functions (each hash function represents a permutation). However, if the number of columns is very large and we cannot afford work which is quadratic on the number of columns, then Locality-Sensitive Hashing is proposed. Locality-Sensitive Hashing aims at reducing the number of pairs of columns that are to be considered by finding quickly many non-similar pairs of columns (and hence eliminating those pairs from further consideration). Briefly, it works as follows: It views the signatures in each column as a column of integers. It partitions the rows of this collection of rows into a number of bands. For each band it hashes the columns into buckets. A pair of columns is a candidate pair if they hash in the same bucket in any band. Tuning on the number of bands allows for a more efficient implementation of this approach.

If the input matrix is not sparse, a random collection of rows serves as a signature. Hamming LSH constructs a series of matrices, each with half as many rows as the previous, by OR-ing together two consecutive rows of the previous matrix.

These algorithms, although very efficient in practice, might still yield *false positives* and *false negatives*, i.e., yield a similarity rule which is false or miss some similarity rules. In [48], a family of algorithms is proposed which is called *Dynamic Miss-Counting (DMC)* that avoid both false positives and false negatives. Two passes over the data are made and the amount of main memory used allows for data of moderate size. The key idea in DMC algorithms is confidence-pruning. For each pair of columns the algorithm counts the number of rows with entries in these columns that disagree and if the count exceeds a threshold they discard this similarity rule.

5.4 Transversals

We point out here the connection between maximal frequent itemsets and transversals [94,80] which are defined as follows: A *hypergraph* is a 0-1 matrix with distinct rows. Each row can be viewed as an *hyperedge* and each column as an element. A *transversal* (a.k.a. *hitting set*) is a set of elements such that each hyperedge contains at least one element from the set. A transversal is *minimal* if no subset of it is a transversal.

Recall that a *frequent itemset* is a subset of the columns such that the number of rows with 1 entries in all those columns is above some support threshold. A *maximal* frequent itemset is a frequent itemset such that no superset is a frequent itemset. Given a support value, an itemset belongs to the negative border iff it is not a frequent itemset and all its subsets are frequent itemsets. The following proposition states the relationship between transversals and maximal frequent itemsets.

Proposition 2. *Let H_{Fr} be the hypergraph of the complements of all maximal frequent itemsets, and let H_{Bd^-} be the hypergraph of all itemsets in the negative border. Then the following holds:*

1. The set of all minimal transversals of H_{Fr} is equal to the negative border.

2. The set of all minimal transversals of H_{Bd^-} is equal to the set of all maximal itemsets.

It is not difficult to prove. A transversal T of H_{Fr} has the property: For each maximal frequent itemset S, the transversal T contains at least one attribute which is not included in this itemset S. Hence the transversal is not a maximal frequent itemset. Hence a minimal transversal belongs to the negative border.

For an example, suppose we have four attributes $\{A, B, C, D\}$ and let all maximal frequent itemsets be $\{\{A, B\}, \{A, C\}, \{D\}\}$, then the hypergraph of complements of those itemsets contains exactly the hyperedges $\{\{C, D\}, \{B, D\}, \{A, B, C\}\}$. All minimal transversals of this hypergraph are $\{\{C, B\}, \{C, D\}, \{A, D\}, \{D, B\}\}$ which is equal to the negative border.

This result is useful because the negative border can be found easier in general and then can be used to retrieve the maximal frequent itemsets.

Transversals have been studied for a long time and hence this connection is useful. In [80] this result is extended in more general framework for which finding maximal frequent itemsets is a subcase. A connection is shown among the three problems of computing maximal frequent itemsets, computing hypergraph transversals and learning monotone boolean functions. This approach as well as the approach taken in [5] has its roots in the use of *diagrams* of models in model theory (see e.g., [30]).

For an excellent detailed exposition of algorithms mentioned in this section see [95].

6 Clustering

There are many different variants of the clustering problem and literature in this field spans a large variety of application areas and formal contexts. Clustering has many applications besides data mining including statistical data analysis, compression, vector quantization. It has been formulated in various contexts such as machine learning, pattern recognition, optimization and statistics. Several efficient heuristics have been invented. In this section, we will review some recent algorithms for massive data and mention some considerations on the quality of clustering.

Informally, the clustering problem is that of grouping together (clustering) similar data items. One approach is to view clustering as a density estimation problem. We assume that in addition to the observed variables for each data item, there is a hidden, unobserved variable indicating the "cluster membership". The data is assumed to be produced by a model with hidden cluster identifiers. A mixture weight $w_i(x)$ is assumed for each data item x to belong to a cluster i. The problem is estimating the parameters of each cluster C_i, $i = 1, \ldots, k$, assuming the number k of clusters is known. The clustering optimization problem is that of finding parameters for each C_i which maximize the likelihood of the clustering given the model.

In most cases, discovery of clusters is based on a distance measure $D(\vec{u}, \vec{v})$, between vectors $\vec{u}, \vec{v}$ (such as the L_p norm) and the three axioms for distance measure hold, i.e., 1. $D(\vec{u}, \vec{u}) = 0$ (reflexivity), 2. $D(\vec{u}, \vec{v}) = D(\vec{v}, \vec{u})$ (symmetry) and 3. $D(\vec{u}, \vec{v}) \leq D(\vec{u}, \vec{z}) + D(\vec{z}, \vec{v})$ (triangle inequality). If the points to be clustered are positioned in some n-dimensional space then Euclidean distance may be used. In general other measures of distances are also useful such as the cosine measure or the *edit distance* which measures the number of inserts and deletes of characters needed to change one string of characters into another.

Most conventional clustering algorithms require space $\Omega(n^2)$ and require random access to the data. Hence recently several heuristics have been proposed for scaling clustering algorithms. Algorithms for clustering usually fall in two large categories *k-median approach* algorithms and *hierarchical approach* algorithms.

6.1 The *k*-Median Approach

A common formulation of clustering is the *k-median* problem: Find k centers in a set of n points so as to minimize the sum of distances from data points to their closest cluster centers. Or, equivalently, to minimize the average distance from data points to their closest cluster centers. The assumptions taken by the classical k-median approach are: 1) each cluster can be effectively modeled by a spherical Gaussian distribution and 2) each data item is assigned to one cluster.

In [18], a single pass algorithm is presented for points in the Euclidean space and is evaluated by experiments. The method used is based in identifying regions of the data that are compressible (*compression set*), other regions that must be maintained in memory (*retained set*) and a third kind of regions that can be completely discarded (*discard set*). The discard set is set of points that are certain to belong to a specific cluster. They are discarded after they are used to compute the statistics of the cluster (such as the number of points, the sum of coordinates, the sum of squares of coordinates). The compression set is set of points that are close to each other so that it is certain that they will be assigned to the same cluster. They are replaced by their statistics (same as for the discard set). The rest of the points that do not belong in either of the two other categories remain in the retained set. The algorithm begins by storing a sample of points (the first points to be read) in main memory and running on them a main memory algorithm (such as k-means [66]). A set of clusters is obtained which will be modified as more points are read into main memory and processed. In each subsequent stage, a main-memory full set of points is processed as follows. 1. Determine if a set of points is (a) sufficiently close to some cluster c_i and (b) unlikely for c_i to "move" far from these points (during subsequent stages) and another cluster come closer. A discard set is decided in this way and its statistics used to update the statistics of the particular cluster. 2. Cluster the rest of the points in main memory and if a cluster is very tight, then replace the corresponding set of points by its statistics; this is a compression set. 3. Consider merging compression sets.

Similar summaries of data as in [18] and a data structure like an R-tree to store clusters are used in [50] to develop a one pass algorithm for clustering points in arbitrary metric spaces.

Algorithms with guaranteed performance bounds include a constant-factor approximation algorithm developed in [57] for the k-median problem. It uses a single pass on the data stream model and requires workspace $\theta(n^{\epsilon})$ for a factor of $2^{O(\frac{1}{\epsilon})}$. Other work includes [12] where the problem is studied on the sliding windows model.

A related problem is the k-center problem (minimize the maximum radius of a cluster) which is investigated in [32] where a single pass algorithm which requires workspace $O(k)$ is presented.

6.2 The Hierarchical Approach

A hierarchical clustering is a nested sequence of partitions of the data points. It starts with placing each point in a separate cluster and merges clusters until it obtains either a desirable number of clusters (usually the case) or a certain quality of clustering.

The algorithm CURE [58] handles large datasets and assumes points in Euclidean space. CURE employs a combination of random sampling and partitioning. In order to deal with odd-shaped clusters, this algorithm selects dispersed points and moves them closer to the centroid of the corresponding cluster. A random sample, drawn from the data set, is first partitioned and cluster summaries are stored in memory in a tree data structure. For each successive data point, the tree is traversed to find the closest cluster to it.

6.3 Similarity Measures

Similarity measures according to which to cluster objects is also an issue of investigation. In [35], methods for determining similar regions in tabular data (given in a matrix) are developed. The proposed measure of similarity is based on the L_p norm for various values for p (non-integral too). It is noticed that on synthetic data, when clustering uses as a distance measure either L_1 or L_2 norms, the quality of the clustering is poorer than when p is between 0.25 and

0.8. The explanation for this is that, for large p, more emphasis is put on the outlier values (*outliers* are points that are isolated, so they do not belong to any cluster), whereas for small p the measure approaches the Hamming distance, i.e., it counts how many values are different. On real data, it is noticed that different values for p bring out different features of the data. Therefore, it seems that p can be used as a useful parameter of the clustering algorithm: set p higher to show full details of the data set, reduce p to bring out unusual clusters in the data. For the technical details to go through, sketching techniques similar to [62] are used to approximate the distances between subtables and reduce the computation. The proposed similarity measure is tested using the k-means algorithm to cluster tabular data.

6.4 Clustering Documents by Latent Semantic Indexing (LSI)

The use of vector space models for information retrieval purposes has been used as early as 1957. The application of SVD in information retrieval is proposed in [37] through the latent semantic indexing technique and it is proven a powerful approach for dimension reduction. The input matrix $\vec{X}$, is a document versus terms matrix. It could be a 0,1-matrix or each entry could be the frequency of the term in this document. Matrix $\vec{X}$ is approximated according to SVD by $\vec{X}_k = \vec{U}_k\vec{T}_k\vec{V}_k^{Tr}$. The choice of k is an issue for investigation. Note that each entry of the matrix $\vec{X}_k$ does not correspond to a term any more, it corresponds to a weighted sum of term measures. The matrix $\vec{V}_k$ represents similarities among documents, e.g., given a document that the user is interested in more documents can be decided that are of interest to this user (even if they do not use exactly the same terms). The matrix $\vec{U}_k$ displays similarities between terms, e.g., given a term, other related terms may be decided (such as the term "car" is related to "automobile" and "vehicle"). The matrix $\vec{X}_k$ may be used for term-document associations, e.g., on a given term, extract documents that contain material related to this term.

Spectral methods –i.e., the use of eigenvectors and singular vectors of matrices–in document information retrieval and the application of SVD through the latent semantic indexing technique are discussed in detail in [73], which is an excellent survey on this direction of research.

6.5 Quality of Clustering

In a clustering algorithm the objective is to find a good clustering but a good clustering is not formally defined. Intuitively the quality of a clustering is assessed by how much similar points are grouped in the same cluster. In [68] the question is posed: how good is the clustering which is produced by a clustering algorithm? As already discussed the k-median clustering may produce a very bad clustering in case the "hidden" clusters are far from spherical. E.g., imagine two clusters, one that is a sphere and a second one is formed at a certain distance around the sphere forming a ring. Naturally the k-median approach will fail to produce a good clustering in this example. A bicriteria measure is proposed therein for assessing the quality of clusters. The dataset is represented as a graph with weights on the edges that represent the degree of similarity between the two vertices (high weight means high similarity). First a quantity which measures the relative minimum cut of a cluster is defined. It is called *expansion* and is defined as the weight of the minimum cut divided by the number of points in the smaller subset among the two that the cut partitions the cluster-graph. It seems however that it is more appropriate to give more importance to vertices with many similar other vertices than to vertices with few similar other vertices. Thus, the definition is extended to capture this observation and the *conductance* is defined where subsets of vertices are weighted to reflect their importance. Optimizing the conductance gives the right clustering in the sphere-ring example. However if we assume the conductance as the measure of quality, then imagine a situation where there are mostly clusters of very good quality and a few points that create clusters of poor quality. In this case the algorithm might create many smaller clusters of medium quality. A second criterion is considered in order to overcome this problem. This criterion is defined as the fraction of the total weight of edges that are not covered by any cluster.

This bicriterion optimization framework is used to measure the quality of several spectral algorithms. These algorithms, though have proven very good in practice, were hitherto lacking a formal analysis.

6.6 Other Algorithms

Popular clustering algorithms in the literature include k-means [66], CLARANS [83], BIRCH [100], DBSCAN [41].

In [87], the drawbacks of random sampling in clustering algorithms (e.g., small clusters might be missed) are avoided by density biased sampling. The goal is to under-sample dense regions and over-sample sparse regions of the data. A memory efficient single pass algorithm is proposed that approximates density biased sampling. In [86], results from [57] are used to develop an algorithm that achieves dramatically better clustering quality than BIRCH although it takes longer to run. In [3], clusters are defined in euclidean space by DNF formulas and performance issues are addressed for data mining applications.

An excellent detailed exposition of algorithms in [58], [18] and [50] can be found in [95]. An excellent survey of the algorithms in [54,3,13,15] is given in [45].

7 Mining the Web

The challenge in mining the web for useful information is the huge size and unstructured organization of data. Search engines, one of the most popular web mining applications, aim to search the web for a specific topic and give to the user the most important web pages on this topic. A considerable amount of research has been done on ranking web pages according to their importance.

Page Rank, the algorithm used by the Google search engine [22] ranks pages according to the page citation importance. This algorithm is based on the observation that usually important pages have many other important pages linking to them. It is an iterative procedure which essentially computes the principal eigenvector of a matrix. The matrix has one nonzero entry for each link from page i to page j and this entry is equal to $1/n$ if page i has n successors (i.e., links to other pages). The intuition behind this algorithm is that page i shares its importance among its successors. Several variants of this algorithm have been developed to solve problems concerning spams and dead ends (pages with no successors). Random jumps from a web page to another may be used to avoid dead ends or a slight modification of the iterative procedure according to which some of the importance is equally distributed among all pages in the beginning.

Hubs and Authorities, based on similar intuition, is an algorithm where web pages are viewed also as sharing their importance among its successors only that there are two different roles assigned to important web pages [72]. They follow the observation that authorities might not link to one another directly but there are hubs that link “collectively” to many authorities. Thus hubs and authorities have this mutually depending relationship: good hubs link to many authorities and good authorities are linked by many hubs. Hubs are web pages that do not contain information themselves but they contain many links to pages with information e.g., a university course homepage. Authorities are pages that contain information about a topic, e.g., a research project homepage. Again the algorithm based on this idea is an iterative procedure which computes eigenvectors of certain matrices. It begins with matrix A similar as the page rank algorithm only that the entries are either 0 or 1 (if there is a link) and its output is two vectors which measure the "authority" and the "hubbiness" of each page. These vectors are the principal eigenvectors of the matrices AA^T and A^TA. Work in [55,77] has shown that the concepts of hubs and authorities is a fundamental structural feature of the web. The CLEVER system [29] builds on the algorithmic framework of hub and authorities.

Other work on measuring the importance of web pages include [1]. Other research directions for mining useful information from the web include [23], where the web is searched for frequent itemsets by a method using features of the algorithm for dynamic itemset counting [21]. Instead of a single deterministic run, the algorithm runs continuously exploring more and more sites. In [19], the extraction of structured data is achieved from information offered by unstructured data on the web. The example used is to search for books in the web starting from a small sample of books from which a pattern is extracted. Based on the extracted patterns more books are retrieved in a iterative manner. Based on the same idea of pattern matching, the algorithm developed in [9] searches the web for communities that share an interest on a topic. The pattern is formed by using words from the anchor text.

More detailed descriptions of the Page Rank and the Hubs and Authorities algorithms can be found in [95]. Also, an elegant formal exposition of spectral methods used for web mining and the connections between this work and earlier work on sociology and citation analysis [39] can be found in [73].

8 Evaluating the Results of Data Mining

As we have seen, for many of the successful data mining algorithms there is no formal analysis as to whether the solution they produce is a good approximation to the problem at hand. Recently a considerable amount of research is focused in developing criteria for such an evaluation.

A line of research is focused in building models for practical situations (like the link structure of the web or a corpus of technical documents) against which to evaluate algorithms. Naturally, the models, in order to be realistic, are shown to display several of the relevant features that are measured in real situations (e.g., the distribution of the number of outgoing links from a web page).

In [88], a model for documents is developed on which the LSI method is evaluated. In this model, each document is built out of a number of different topics (hidden from the retrieval algorithm). A document on a given topic is generated by repeating a number of times terms related to the topic according to a probability distribution over the terms. For any two different topics there is a technical condition on the distributions that keeps the topics "well-separated". The main result is that on this model, the k-dimensional subspace produced by LSI defines with high probability very good clusters as intended by the hidden parameters of the model. In this paper, it is also proposed that if the dimensionality after applying LSI is too large, then random projection can be used to reduce it and improve the results. Other work with results suggesting methods for evaluating spectral algorithms include [10].

Models for the web graph are also developed. The web can be viewed as a graph with each page being a vertex and an edge exists if there is a link pointing from one web page to another. Measurements on the web graph [76,77] have shown that this graph has several characteristic features which play a major role in the efficiency of several known algorithms for searching the web. In that context, the web graph is a power-law graph, which means roughly that the probability that a degree is larger than d is proportional to $d^{-\beta}$ for some $\beta > 0$. Models for power-law graphs are developed in [42],[7], [78].

A technique for automatically evaluating strategies which find similar pages on the web is presented in [60]. A framework for evaluating the results of data mining operations according to the utility of the results in decision making is formalized in [74] as an economically motivated optimization problem. This framework leads to interesting optimization problems such as the segmentation problem which is studied in [75]. Segmentation problems are related to clustering.

9 Conclusion

We surveyed some of the formal techniques and tools for solving problems on the data stream model and on similar models where there are constraints on the amount of main memory used and a few number of passes through the data are allowed because access is too costly. We also presented some of the most popular algorithms that are proposed in the literature for data mining applications. We provided reference for further reading, with some good surveys and tutorials in the end of each section. As the field is a rapidly evolving area of research with many diverging applications, the exposition here is meant to serve as an introduction to approximation algorithms with storage constraints and their applications.

Among topics that we did not mention are *Privacy preserving data mining*, *Time Series Analysis*, *Visualization* of Data Mining results, *Bio-informatics*, *Semistructured data and XML.*

Acknowledgements. Thanks to Chen Li and Ioannis Milis for reading and providing comments in an earlier version of this chapter.

References

1. S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *VLDB*, 2002.
2. F. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *KDD*, 2004.
3. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 1998.
4. R. Agrawal, T. Imielinski, and A. Swami. Mining associations between sets of items in massive databases. In *SIGMOD*, pages 207–216, 1993.
5. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining.* 1996.
6. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.
7. W. Aiello, F. Chung, and L. Lu. A random graph model for power law graphs. In *STOC*, 2000.
8. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating frequency moments. In *STOC*, pages 20–29, 1996.
9. N. AlSaid, T. Argyros, C. Ermopoulos, and V. Paulaki. Extracting cyber communities through patterns. In *SDM*, 2003.
10. Y. Azar, A. Fiat, A. Karlin, F. McSherry, and J. Saia. Spectral analysis of data. In *STOC*, pages 619–636, 2001.
11. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, 2002.
12. B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS*, 2003.
13. J. Banfield and A. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, 49:803–821, 1993.
14. Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. In *STOC*, 2001.

15. A. Ben-Dor and Z. Yakhini. Clustering gene expression patterns. In *RECOMB*, 1999.
16. J.O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer Verlag, 1985.
17. A. Borodin, R. Ostrovsky, and Y. Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. In *STOC*, 1999.
18. P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *KDD*, 1998.
19. S. Brin. Extracting patterns and relations from the world-wide web., 1998.
20. S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD*, pages 265–276, 1997.
21. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD*, pages 255–264, 1997.
22. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW7/Computer Networks*, pages 107–117, 1998.
23. S. Brin and L. Page. Dynamic data mining: Exploring large rule space by sampling., 1998.
24. A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, pages 21–29, 1997.
25. A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC*, 1998.
26. A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the web. In *Sixth International World Wide We Conference*, pages 391–404, 1997.
27. H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. available at: http://www.cwi.nl/ rdewolf, 1999.
28. R. Canetti, G. Even, and O. Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53:17–25, 1995.
29. S. Chakrabarti, B. Dom, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Experiments in topic distillation. In *SIGIR workshop on hypertext information retrieval*, 1998.
30. C.C. Chang and H.J. Keisler. *Model Theory.* North Holland, Amsterdam, 1990.
31. M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *PODS*, pages 268–279, 2000.
32. M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *STOC*, pages 626–635, 1997.
33. E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and Systems Sciences*, 55:441–453, 1997.
34. E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. In *TKDE 13(1) 2001 and also in ICDE*, pages 64–78, 2000.
35. G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. In *ICDE*, 2002.
36. P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for monte carlo estimation. In *FOCS*, pages 142–149, 1995.
37. S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *The American Society for Information Science*, 41(6):391–407, 1990.
38. A. Dobra, M. Garofalakis, and J. Gehrke. Sketch-based multi-query processing over data streams. In *EDBT*, 2004.
39. L. Egghe and R. Rousseau. *Introduction to Informetrics.* Elsevier, 1990.

40. L. Engebretsen, P. Indyk, and R. O'Donnell. Derandomized dimensionality reduction with applications. In *SODA*, 2002.
41. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knoweledge Discovery and Data Mining*, page 226, 1996.
42. A. Fabrikant, E. Koutsoupias, and C. H. Papadimitriou. Heuristically optimized trade-offs: A new paradigm for power laws in the internet. In *STOC*, 2002.
43. C. Faloutsos. Indexing and mining streams. In *SIGMOD*, 2004.
44. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB*, 1998.
45. D. Fasulo. An analysis of recent work on approximation algorithms. Technical Report 01-03-02, University of Washington, Dept. of Computer science and Engineering, 1999.
46. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate l1-difference for massive data streams. In *FOCS*, 1999.
47. W. Feller. *An Introduction to Probability Theory and Its Applications.* John Wiley, New York, 1968.
48. S. Fujiwara, J. D. Ullman, and R. Motwani. Dynamic miss-counting algorithms: Finding implication and similarity rules with confidence pruning. In *ICDE*, pages 501–511, 2000.
49. S. Gangulya, M. Garofalakis, and R. Rastogi. Sketch-based processing data streams join aggregates using skimmed sketches. In *EDBT*, 2004.
50. V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French. Clustering large datasets in arbitrary metric spaces. In *ICDE*, pages 502–511, 1999.
51. M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. In *VLDB*, 2002, also available at: http://www.bell-labs.com/˜ minos.
52. P. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *SODA*, pages S909–S910, 1999.
53. P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 281–291, 2001.
54. D. Gibson, J. M. Kleinberg, and P. Raghavan. Two algorithms for nearest neighbor search in high dimensions. In *STOC*, volume 8(3-4), 1997.
55. D. Gibson, J. M. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *ACM Conference on Hypertext and Hypermedia*, volume 8(3-4), 1998.
56. A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.
57. S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS*, 2000.
58. S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD*, 1998.
59. D.J. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining (Adaptive computation and machine learning).* MIT Press, 2001.
60. T. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Similarity search on the web: Evaluation and scalable considerations. In *11th International World Wide Web Conference*, 2002.
61. M. R. Henzinger, P. Raghavan, and S. Rajagopalan. *Computing on data streams.* available at: http://www.research.digital.com/SRC/, 1998.

62. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.
63. P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *FOCS*, 2001.
64. P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB*, pages 363–372, 2000.
65. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.
66. A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.
67. W.B. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.
68. R. Kannan, S. Vempala, and A. Vetta. On clusterings - good, bad and spectral. In *FOCS*, pages 367–377, 2000.
69. A.R. Karlin, M.S. Manasse, L. Rodolph, and D.D. Sleator. Competitive snoopy caching. In *STOC*, pages 70–119, 1988.
70. M.J. Kearns and U.V. Vazirani. *An introduction to comoputational learning theory.* MIT Press, 1994.
71. D. Kifer, J. Gehrke, C. Bucila, and W. White. How to quickly find a witness. In *PODS*, 2003.
72. J. Kleinberg. Authoritative sources in a hyperlinked environment. *J.ACM*, 46(5):604–632, 1999.
73. J. Kleinberg and A. Tomkins. Applications of linear algebra in information retrieval and hypertext analysis. In *PODS*, pages 185–193, 1999.
74. J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2(4):311–324, 1998.
75. J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. Segmentation problems. In *STOC*, pages 473–482, 1998.
76. S.R. Kumar, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: experiments and models. In *International World Wide Web Conference*, pages 309–320, 2000.
77. S.R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling emerging cybercommunities automatically. In *International World Wide Web Conference*, volume 8(3-4), 1999.
78. S.R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *FOCS*, pages 57–65, 2000.
79. E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.
80. H. Mannila and H. Toivonen. On an algorithm for finding all interesting sentences. In *Cybernetics and Systems, Volume II, The Thirteenth European Meeting on Cybernetics and Systems Research*, pages 973 – 978, 1996.
81. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
82. J.L. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
83. R.T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, pages 144–155, 1994.
84. N. Nisan. Pseudorandom generators for pseudorandom computations. In *STOC*, pages 204–212, 1990.
85. J.P. Nolan. *An introduction to stable distributions.* http://www.cas.american.edu/ jpnolan/chap1.ps.

86. L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *ICDE*, 2002.
87. C. Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. In *SIGMOD*, pages 82–92, 2000.
88. C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. *JCSS*, 61(2):217–235, 2000.
89. J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD*, pages 175–186, 1995.
90. M. Saks and X. Sun. Space lower bounds for distance approximation in the data stream model. In *STOC*, 2002.
91. L. Schulman and V.V. Vazirani. Majorizing estimators and the approximation of ♯p-complete problems. In *STOC*, pages 288–294, 1999.
92. C. Silverstein, S. Brin, R. Motwani, and J. D. Ullman. Scalable techniques for mining causal structures. In *Data Mining and Knowledge Discovery 4(2/3)*, pages 163–192, 2000.
93. STREAM. Stanford stream data management project. http://www-db.stanford.edu/stream.
94. H. Toivonen. Sampling large databases for association rules. In *VLDB*, pages 134–145, 1996.
95. J. D. Ullman. Lecture notes on data mining. available at: http://www-db.stanford.edu/˜ ullman/cs345-notes.html, 2000.
96. V.N. Vapnik. *Statistical learning theory.* John Wiley, 1998.
97. V.V. Vazirani. *Approximation algorithms.* Springer, 2001.
98. D.E Vengroff and J.S. Vitter. I/o efficient algorithms and environments. *Computing Surveys*, page 212, 1996.
99. J. Vitter. Random sampling with a reservoir. *ACM Trans. on Mathematical Software*, 11(1):37–57, 1985.
100. T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *SIGMOD*, pages 103–114, 1996.

A Survey of Approximation Results for Local Search Algorithms

Eric Angel

LaMI, CNRS-UMR 8042, Université d'Évry Val-d'Essonne, 91025 Evry, France
angel@lami.univ-evry.fr

Abstract. In this chapter we review the main results known on local search algorithms with worst case guarantees. We consider classical combinatorial optimization problems: satisfiability problems, traveling salesman and quadratic assignment problems, set packing and set covering problems, maximum independent set, maximum cut, several facility location related problems and finally several scheduling problems. A replica placement problem in a distributed file systems is also considered as an example of the use of a local search algorithm in a distributed environment.

For each problem we have provided the neighborhoods used along with approximation results. Proofs when too technical are omitted, but often sketch of proofs are provided.

1 Introduction

The use of local search in combinatorial optimization reaches back to the late 1950s and early 1960s. It was first used for the traveling salesman problem and since then it has been applied to a very broad range of problems [1]. While the basic idea is very simple, it has been considerably used and extended in more elaborate algorithms such as simulated annealing and taboo search. Local search algorithms are also often hybridised with other resolution methods such as genetic algorithms. Such methods are commonly referred under the term of metaheuristics [19,93].

In this survey we are concerned with "pure" local search algorithms which provide solutions with some guarantee of performance. While the previous metaheuristics may be very efficient in practice for obtaining near to optimal solutions for a large class of combinatorial optimization problems, one lacks theoretical results concerning the quality of solution obtained in the worst case. Indeed, theoretical works during the last decade mainly addressed the problem of the computational difficulty of finding local optima solutions [98,104], regardless of the quality achieved. For example in 1997 Yannakakis [104] in his survey mentioned that very little work analyzed performance of local search. However, recently more and more results involving approximation results using local search algorithms appeared, and therefore we feel that time has come to make a survey of known results. We hope it will motivate further study in this area and give insight on the power and limitations of the local search approach for solving combinatorial optimization problems.

In this chapter we consider only the standard approximation ratio [11,100]. For other approximation results using the differential approximation ratio, the reader is referred to the book of Monnot, Paschos and Toulouse [83]. This chapter is organized as follows. In Section 2 we introduce local search algorithms, and we discuss convergence issues of these algorithms. In Section 2.3 we consider local search algorithms with respect to polynomially solvable problems. Section 3 is devoted to satisfiability problems. Several results about graph and hypergraph coloring problems are also presented, since they are corollaries of the results obtained for satisfiability problems. In Section 4 we consider the famous traveling salesman problem. Section 5 is devoted to the quadratic assignment problem which is a generalization of the traveling salesman problem. Several results for other combinatorial optimization problems, such as the traveling salesman problem and the graph bipartitioning problem, since they are particular cases of the quadratic assignment problem, are also presented. In Section 6 we consider set packing and maximum independent set problems. In Section 7 we consider the set covering problem. Section 8 is devoted to the maximum cut problem. In Section 9 we consider several facility location related problems, and in Section 10 we consider several classical scheduling problems. The next sections consider less known combinatorial optimization problems: Section 11 is devoted to the minimum label spanning tree problem, whereas Section 12 is devoted to a replica placement problem in a distributed file systems. Finally in Section 13 we resume the main approximation results obtained in a single table, and suggest some investigations for further research.

2 Local Search Algorithms

2.1 Introduction

Let us consider an instance $\mathcal{I}$ of a combinatorial optimization problem. The instance $\mathcal{I}$ is characterized by a set $\mathcal{S}$ of feasible solutions, and a cost function C such that $C(s) \in \mathbb{N}$. Assuming a minimization problem, the aim is to find a *global optimal* solution, i.e. a solution $s^* \in \mathcal{S}$ such that $C(s^*) \leq C(s)$ for all $s \in \mathcal{S}$. In the case of a maximization problem, one look for a solution $s^* \in \mathcal{S}$ such that $C(s^*) \geq C(s)$ for all $s \in \mathcal{S}$.

To use a local search algorithm one needs a *neighborhood*. The neighborhood $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ associates to each solution $s \in \mathcal{S}$ a set $\mathcal{N}(s) \subseteq \mathcal{S}$ of neighboring solutions. A neighboring solution of s is traditionally obtained by performing a small transformation on s. The size $|\mathcal{N}|$ of the neighborhood $\mathcal{N}$ is the cardinality of the set $\mathcal{N}(s)$ if this quantity does not depend of the solution s, which is very often the case in practice.

The generic local search algorithm is depicted in algorithm 1. It is an iterative improvement method in which at each step one tries to improve the current solution by looking at its neighborhood. If at each step the current solution is replaced by a best (resp. any better) solution in its neighborhood, one speak of deepest (resp. first descent) local search.

Algorithm 1 Generic local search algorithm

```
Let $s \in \mathcal{S}$ be an initial solution
while there is a solution $x \in \mathcal{N}(s)$ that is strictly better than the current solution $s$ do
    $s \leftarrow x$ (i.e. we replace the current solution $s$ by a strictly better solution in its neighborhood $\mathcal{N}(s)$)
end while
return $s$
```

A *local optimum* solution $s_{loc} \in \mathcal{S}$ is a solution which is better than all its neighboring solutions, i.e. $C(s_{loc}) \leq C(s)$ for all $s \in \mathcal{N}(s_{loc})$ for a minimization problem, and $C(s_{loc}) \geq C(s)$ for all $s \in \mathcal{N}(s_{loc})$ for a maximization problem. The local search algorithm always ends at a local optimum solution. Notice that a global optimum solution is always a local optimum, but the converse is in general not true.

Despite their simplicity, we will see that local search algorithms can lead to approximation algorithms for a large class of combinatorial optimization problems. A local search algorithm is said to be a ρ approximation algorithm if *any* local optimum solution satisfies $C(s_{loc})/C(s^*) \leq \rho$ (resp. $C(s^*)/C(s_{loc}) \leq \rho$) for a minimization (resp. maximization) problem, with s^* a global optimal solution. Therefore the approximation ratio ρ is always greater than 1. The closer to 1 is ρ, the better is the approximation. Notice that some authors consider the inverse ratio $1/\rho$ instead of ρ for maximization problems. In case $\rho = 1$ the neighborhood is said to be *exact*.

2.2 Convergence Issue

To be a polynomial time algorithm the local search must find a local optimum within a polynomial number of iterations, and each iteration must take a polynomial time.

The second condition is always met if the neighborhood has a polynomial size. However recently there has been some interest in exponential sized neighborhoods which can be searched in polynomial time using different techniques (dynamic programming for instance), see [2,51] for a survey. However, as far as we know, despite interesting experimental results there is no approximation algorithm based on such neighborhoods yet (excepted the paper of Boykov, Veksler and Zabih [20] which is considered in chapter X of this book).

The first condition is sometimes easy to check. The standard argument is as follows. If the cost function is integral, non negative, and it is allowed to take only values bounded by a polynomial in the size of the instance, assuming we have a minimization (resp. maximization) problem, then since at each iteration the cost value of the current solution must decrease (resp. increase) by at least one unit, it means that a local optimum will be reached in a polynomial number of iterations.

Ausiello and Protasi [12] defined a class of optimization problems called GLO (Guaranteed Local Optima) using such assumptions. They also show that GLO can be seen as the core of APX, the class of problems that are approximable in polynomial time, in the sense that all problems in APX either belong to GLO or may be reduced to a problem in GLO by means of approximable preserving reductions.

However if the cost function can take exponential values, a local search algorithm may reach a local optimum only after an exponential number of iterations. Johnson, Papadimitriou and Yannakakis have defined in [66] the PLS (polynomial local search) complexity class in order to study the intrinsic complexity of finding a local optimum with respect to a given neighborhood (not by using necessarily a local search algorithm). It has been shown that several problems are PLS-complete [66,94,96,103].

However recently Orlin, Punnen and Schulz [85] have introduced the concept of ϵ-local optimality and showed that for a large class of combinatorial optimization problems, an ϵ-local optimum can be identified in time polynomial in the problem size and $1/\epsilon$ whenever the corresponding neighborhood can be searched in polynomial time, for $\epsilon > 0$. For a minimization problem, s is an ϵ-local optimum if $\frac{C(s)-C(s')}{C(s')} \leq \epsilon$, for all $s' \in \mathcal{N}(s)$. An ϵ-local optimum has nearly the properties of a local optimum, however one should point out that an ϵ-local optimum is not necessarily close to a true local optimum solution.

2.3 Polynomially Solvable Problems

Before considering NP-hard problems in the sequel, one may wonder what the behavior of local search is on polynomially solvable problems.

Following Yannakakis [104] one can say that for linear programming the simplex algorithm can be view as a local search. Indeed, a vertex of the polytope of feasible solutions is not an optimal solution if and only if it is adjacent to another vertex of the polytope with better cost. Therefore the adjacency vertices of the polytope defines an exact neighborhood. Algebraically it corresponds to exchanging a variable of the basis for another variable outside the basis.

The weighted matching problem in a graph is a well known polynomially solvable problem. The fastest algorithm known is due to Gabow [44] with a running time of $\mathcal{O}(|V|\,|E| + |V|^2 \log |V|)$. However this running time is sometimes too large for real world instances. Recently Drake and Hougardy [36] have proposed a local search algorithm in linear time for the weighted matching problem in graph with a performance ratio of 3/2. For the maximum matching problem, notice that a matching is not maximum if and only if it has an augmenting path. Therefore, if we say that two matchings are neighbors if their symmetric difference is a path, then we have an exact neighborhood for the maximum cardinality matching problem. The neighborhood used by Drake and Hougardy is based on such augmenting structures.

For the minimum spanning tree problem, a non optimal tree can be improved by adding an edge to the tree and removing another edge in the (unique) cycle formed. Thus the neighborhood in which 2 spanning trees are neighbor if one can be obtained from the other by exchanging one edge for another is exact. It can be shown that after at most a polynomial number of iterations the local search algorithm converges [62].

More general results can be obtained using the framework of matroids. Recall that a *matroid* M is a finite set $E(M)$ together with a subset $\mathcal{J}(M) \subseteq E(M)$ that satisfies the following properties:

1. $\emptyset \in \mathcal{J}(M)$
2. $X \subset Y \in \mathcal{J}(M) \Longrightarrow X \in \mathcal{J}(M)$
3. $X \in \mathcal{J}(M)$, $Y \in \mathcal{J}(M)$, $|Y| > |X| \Longrightarrow \exists e \in Y \setminus X$ such that $X + e \in \mathcal{J}(M)$

The set $\mathcal{J}(M)$ is called the set of *independent sets* of M.

Given a weight function C we consider the problem of finding maximum-weight independent sets S_k of cardinality k. This problem can be solved efficiently in polynomial time using the weighted greedy algorithm (see for example [77]). As an example consider a graph $G(V, E)$, let $E(M) = E$ and let $\mathcal{J}(M)$ be the set of forests (set of edges containing no cycle) of G. Then M is called the *graphic matroid* of G, and obtaining a maximum (or minimum) weighted spanning tree of G can be obtained using a greedy algorithm known as Kruskal's algorithm.

Let Δ denotes the symmetric difference of sets, i.e. $S \Delta S' = (S \setminus S') \cup (S' \setminus S)$. Let us consider the following local search algorithm: Choose any $S \in \mathcal{J}(M)$, such that $|S| = k$. While there exists $S' \in \mathcal{J}(M)$ with $|S'| = k$ and $|S \Delta S'| = 2$ and $C(S') > C(S)$: Let $S := S'$. Interestingly it can be shown that if M is a matroid, then this swapping algorithm finds a maximum-weight independent set of cardinality k [77].

Rardin and Sudit [91,92] introduced a new *paroid* structure designed to provide a canonical format for formulating combinatorial optimization problems together with a neighborhood structure. They introduced a generic local optimization scheme over paroids which unifies some well known local search heuristics such as the Lin-Kernighan heuristic for the traveling salesman problem.

3 Satisfiability Problems

Local search is widely used for solving a wide range of satisfaction problems. In this section we review the main results known.

3.1 Definitions and Problems

Let us consider a set of n boolean variables x_j, for $1 \leq j \leq n$. We note $\overline{truth} = false$ and $\overline{false} = truth$. A literal is either a boolean variable x_j or its negation $\overline{x_j}$. In a constraint satisfaction problem, the input is given by a set of m boolean clauses C_i, $1 \leq i \leq m$. In the sequel, we shall assume that no clause contains both a literal and its negation. The objective is to find an assignment of truth values to the boolean variables (a truth assignment) in order to satisfy the maximum number of clauses.

According to the structure and restrictions over the constraints, we obtain several problems. In the MAX k-SAT satisfiability problem, each clause C_i is a disjunction of exactly k literals (in the literature one sometimes assume *up to* k literals per clause instead of exactly k). If each clause can contain an arbitrary number of literals, this problem is denoted by MAX SAT. In the MAX k-CCSP conjunctive constraint satisfaction problem, each clause C_i is a conjunction of k literals.

In the MAX CSP constraint satisfaction problem we have a set of variables V. The domain of each variable is $K = \{1, \ldots, k\}$. A constraint between two variables u and v, denoted by $R(u, v)$, is a binary relation on $K \times K$, which defines the pair of values that can be assigned to u and v simultaneously. An assignment is a function $f : V \to K$. A constraint $R(u, v)$ is said to be satisfied if $(f(u), f(v)) \in R(u, v)$. A constraint $R(u, v)$ is said to be r-consistent $(1 \leq r \leq k)$ if and only if for every value $i_1 \in K$ there exists at least r consistent values $i_2 \in K$ such that $(i_1, i_2) \in R(u, v)$ An instance is said to be r-consistent if all its constraints are r-consistent. In the MAX CSP(k, r) constraint satisfaction problem, all the instances have domain size k and are r-consistent. Notice that MAX k-CUT (see Section 8) is equivalent to MAX CSP$(k, k-1)$ with all not-equal constraints. An instance of a constraint satisfaction problem can be represented by a constraint (multi) graph $G = (V, E)$. There is an edge between u and v for each constraint $R(u, v)$. We denote by $d(v)$ the degree of vertex $v \in V$, i.e. the number of constraints involving this variable, and we denote by Δ the maximum degree of graph G.

All these problems are NP-hard. The decision problem SAT associated with MAX SAT, i.e. to determine if there exists a truth assignment such that all the clauses are satisfied, was the first example of an $\mathcal{NP}$-complete problem [45]. The above problems can be generalized by assigning weights to clauses. For example in the MAX W-SAT problem each clause has a weight, and the objective is to maximize the sum of the weights of the satisfied clauses.

3.2 Oblivious Local Search for MAX k-SAT

Papadimitriou [86] showed that randomized local search could solve the decision problem 2-SAT in polynomial time. More precisely, consider the following algorithm: "Start with any truth assignment. While there are unsatisfied clauses, pick one and flip a random literal in it.". Considering the Hamming distance of the current assignment from an optimal solution satisfying all clauses, one can observes that at each step this distance is at least as likely to be decreased as to be increased, and is bounded up by n the number of variables. Therefore viewing this process as a gambler's ruin problem with a reflecting barrier at most n, one can proves that this algorithm after $\mathcal{O}(n^2)$ number of steps will find a satisfying truth assignment, if one exists, with probability arbitrarily close to one.

Dantsin et al. [34] proposed an exact deterministic local search algorithm for MAX k-SAT in exponential time $(2-\frac{2}{k+1})^n$ up to a polynomial factor. The key idea is the use of covering codes to determine the set of initial solutions on which local search is applied. Recently Hirsch [58] has proposed a randomized $(1-\epsilon)$-approximation algorithm. In the following we consider only polynomial time deterministic algorithms.

Let a solution s of a satisfiability problem be represented as a 0-1 vector $s=(s_1,\ldots,s_n)$ with the meaning that $s_i=1$ (resp. $s_i=0$) if x_i is given the value *true* (resp. *false*). One can then define a neighborhood $\mathcal{N}_d$ in the following way: $s' \in \mathcal{N}_d(s) \iff d_H(s,s') \leq d$, with d a constant and d_H the Hamming distance between s and s', i.e. $d_H(s,s') = |\{l : \ s_l \neq s'_l\}|$. Following Alimonti [3,4,5] we will speak of d-bounded neighborhood for $\mathcal{N}_d$. We use flip to mean 1-neighborhood. In the flip (resp. d-bounded neighborhood) one is allowed to change the value of only a single variable (resp. up to d variables) to obtain a neighboring solution. The size of the d-bounded neighborhood is $\binom{n}{d}$, which is a polynomial in the size of the instance since d is a fixed constant, leading to a polynomial time local search algorithm.

Theorem 1 (Hansen and Jaumard, 1990 [55]). *The local search algorithm with the flip neighborhood is a polynomial time $(k{+}1)/k$-approximation algorithm for* MAX k-SAT*, and this ratio is tight.*

Proof. Without loss of generality, we can assume that in the local optimum each variable is assigned the value *true*. If it is not the case, by putting $x'_i = \overline{x_i}$ if $x_i \leftarrow false$, and $x'_i = x_i$ if $x_i \leftarrow true$ in the local optimum, we obtain an equivalent instance for which the assumption is verified.

Let δ_i the variation of the number of satisfied clauses when variable x_i is flipped. Since the assignment is a local optimum, flipping any variable decreases the number of satisfied clauses, i.e. $\delta_i \leq 0$, for $1 \leq i \leq n$. Let cov_s the subset of clauses that have exactly s literals matched by the current assignment, and $cov_s(l)$ the number of clauses in cov_s that contain literal l. We have $\delta_i = -cov_1(x_i) + cov_0(\overline{x_i})$. Indeed, when x_i is flipped from *true* to *false* one looses the clauses that contain x_i as the single matched literal, i.e. $cov_1(x_i)$, and gains the clauses that have no matched literal and that contain $\overline{x_i}$, i.e. $cov_0(\overline{x_i})$. After summing over all variables, we obtain $\sum_{i=1}^n \delta_i \leq 0$, thus

$$\sum_{i=1}^{n} cov_0(\overline{x_i}) \leq \sum_{i=1}^{n} cov_1(x_i). \tag{1}$$

By using the following equalities: $\sum_{i=1}^n cov_1(x_i) = |cov_1|$ and $\sum_{i=1}^n cov_0(\overline{x_i}) = k|cov_0|$, which can be easily verified, we obtain from (1) that $k|cov_0| \leq |cov_1| \leq m_{loc}$. Let m_{loc} be the number of satisfied clauses at a local optimum with the flip neighborhood. Therefore $m = m_{loc} + |cov_0| \leq (1+\frac{1}{k})m_{loc} = \frac{k+1}{k}m_{loc}$.

Of course when flipping a variable the net increase in the number of satisfied clauses is at least one, and therefore the total numbers of moves before reaching a local optimum is bounded up by m, the total number of clauses, and is therefore polynomial. Moreover each iteration of the local search takes a polynomial time. □

Battiti and Protasi give the following intuitive explanation [15]: "if there are too many clauses in cov_0, because each of them has k unmatched literals, there will be at least one variable whose flipping will satisfy so many of these clauses to lead to a net increase in the number of satisfied clauses."

Can we improve the approximation ratio of Theorem 1 by using a larger neighborhood than flip? Khanna, Motwani, Sudan and Vazirani [70,71] have proved that the performance ratio for any oblivious local search algorithm with a d-bounded neighborhood for MAX 2-SAT is $3/2$ for any $d = o(n)$.

3.3 Non-oblivious Local Search for MAX k-SAT

In *non-oblivious* local search, independently introduced in [3] and in [70], one uses a different objective function than the original one to guide the search. As we shall see, this technique allows to achieve better performance ratio for some problems approximable by means of oblivious local search, and allows to approximate problems not approximable with the oblivious local search.

For the MAX k-SAT problems we consider, it means that rather considering the number of satisfied clauses we use a different criterion in order to guide the local search algorithm towards better local optima.

Recall that the d-neighborhood of a given truth assignment is defined as the set of all assignments that can be obtained by changing the values of at most d variables.

Theorem 2 (Khanna, Motwani, Sudan and Vazirani, 1994 [70,71]). *There exists a non-oblivious local search algorithm with the flip neighborhood, which is a polynomial time $2^k/(2^k-1)$-approximation algorithm for* MAX k-SAT.

Proof. We first give the proof for the MAX 2-SAT problem. As in the proof of Theorem 1 we assume that in the local optimum each variable is assigned the value *true.*

The non-oblivious objective function f_{NOB} is a weighted linear combination of the number of clauses with one and two matched literals: $f_{NOB} = \frac{3}{2}|cov_1| + 2|cov_2|$ (notice that the oblivious objective function is simply $f_{OB} = |cov_1| + |cov_2|$).

Let δ_i (resp. $(\delta|cov_1|)_i$ and $(\delta|cov_2|)_i$) the variation of the objective function f_{NOB} (resp. $|cov_1|$ and $|cov_2|$) when variable x_i is flipped. We have, $\delta_i = \frac{3}{2}(\delta|cov_1|)_i + 2(\delta|cov_2|)_i$, with $(\delta|cov_1|)_i = -cov_1(x_i) + cov_0(\overline{x_i}) + cov_2(x_i) - cov_1(\overline{x_i})$ and $(\delta|cov_2|)_i = cov_1(\overline{x_i}) - cov_2(x_i)$. For a local optimum relative to function f_{NOB} we have $\delta_i \le 0$, for $1 \le i \le n$. After summing over all variables, we obtain $\sum_{i=1}^n \delta_i \le 0$, thus

$$\frac{3}{2}cov_0(\overline{x_i}) + \frac{1}{2}cov_1(\overline{x_i}) \le \frac{1}{2}cov_2(x_i) + \frac{3}{2}cov_1(x_i). \qquad (2)$$

Using the following easily derived equalities $\sum_{i=1}^{n} cov_1(x_i) = \sum_{i=1}^{n} cov_1(\overline{x_i}) = |cov_1|$, $\sum_{i=1}^{n} cov_0(\overline{x_i}) = 2|cov_0|$, and $\sum_{i=1}^{n} cov_2(x_i) = 2|cov_2|$, we obtain from (2) $3|cov_0| \le |cov_2| + |cov_1|$, meaning that the number of satisfied clauses is larger than three times the number of unsatisfied ones. Since $|cov_0|+|cov_1|+|cov_2| = m$ the total number of clauses, this implies that $|cov_0| \le m/4$, that is $m_{loc} \ge \frac{3}{4}m$.

In the general case we have k literals per clause, and

$$(\delta|cov_j|)_i = -cov_j(x_i) + cov_{j-1}(\overline{x_i}) + cov_{j+1}(x_i) - cov_j(\overline{x_i}),$$

for $1 \le j \le k-1$, and

$$(\delta|cov_k|)_i = cov_{k-1}(\overline{x_i}) - cov_k(x_i), \quad (\delta|cov_0|)_i = cov_1(x_i) - cov_0(\overline{x_i}).$$

The objective function is $f_{NOB} = \sum_{j=0}^{k} c_j|cov_j|$, with coefficients c_j, $0 \le j \le k$, to be determined latter. Then $(\delta f_{NOB})_i = \sum_{j=0}^{k} c_j(\delta|cov_j|)_i \le 0$ for any local optimum. After some simple calculus one gets $(\delta f_{NOB})_i = \Delta_1 cov_0(\overline{x_i}) + \sum_{j=1}^{k-1}(\Delta_{j+1}cov_j(\overline{x_i}) - \Delta_j cov_j(x_i)) - \Delta_k cov_k(x_i)$, with $\Delta_j = c_j - c_{j-1}$, $1 \le j \le k$. By using $\sum_{i=1}^{n} cov_j(x_i) = j|cov_j|$, $\sum_{i=1}^{n} cov_j(\overline{x_i}) = (k-j)|cov_j|$ and $\sum_{i=1}^{n}(\delta f_{NOB})_i \le 0$, one gets $k\Delta_k|cov_k| + \sum_{j=1}^{k-1}(j\Delta_j - (k-j)\Delta_{j+1})|cov_j| \ge k\Delta_1|cov_0|$. If we put $\Delta_j = \frac{1}{(k-j+1)\binom{k}{j-1}} \sum_{l=0}^{k-j} \binom{k}{l}$ the coefficient of each term on the above left hand side inequality is unity, so one gets $\sum_{j=1}^{k} |cov_j| = m_{loc} \ge (2^k-1)|cov_0| = (2^k-1)(m - m_{loc})$, thus $m_{loc} \ge (1 - \frac{1}{2^k})m$. □

It can be shown that theorem 2 cannot be improved by using a different weighted linear combination of $|cov_1|$ and $|cov_2|$.

As an example of the superiority of non-oblivious local search over the traditional local search, consider the following set of clauses (taken from [15]): $C_1 = \overline{x_1} \vee \overline{x_2} \vee x_3$, $C_2 = \overline{x_1} \vee \overline{x_2} \vee x_4$, $C_3 = \overline{x_1} \vee \overline{x_2} \vee x_5$ and $C_4 = \overline{x_3} \vee \overline{x_4} \vee \overline{x_5}$. The assignment $\{x_i \leftarrow \mathit{true} \mid 1 \le i \le 5\}$, in which all variables get the value *true*, is a local optimum with respect to the oblivious objective function. It satisfies the first three clauses C_1, C_2 and C_3. However using the non-oblivious objective function $f_{NOB} = \frac{7}{3}|cov_1| + 3|cov_2| + \frac{10}{3}|cov_3|$, it is no more a local optimum since setting $x_1 \leftarrow \mathit{false}$ allows to increase f_{NOB} from 7 to 9. Then at the next iteration we can set $x_3 \leftarrow \mathit{false}$ to get the assignment $\{x_2, x_4, x_5 \leftarrow \mathit{true},\ x_1, x_3 \leftarrow \mathit{false}\}$ which satisfies all the clauses and is a global optimum.

Interestingly, Battiti and Protasi have reported that this approach also improves the experimental results obtained on benchmark instances [14]. Once a local optimum has been found using the non-oblivious local search, an oblivious local search is started from that local optimum.

It is not possible to improve the ratio given in Theorem 2 since it has been shown that MAX k-SAT is not approximable within $2^k/(2^k-1) - \epsilon$ for any $\epsilon > 0$ and $k \ge 3$ [57].

3.4 Non-oblivious Local Search for MAX *k*-CCSP

Let S_j be the set of clauses where j literals are false in a truth assignment T. The non oblivious objective function used within the local search is defined by

$f_{NOB} = \sum_{j=0}^{k} L_j|S_j|$, with $L_j = (1 + kL_{j+1} - (k - j - 1)L_{j+2})/(j + 1)$ for $0 \le j \le k - 1$, and $L_j = 0$ for $j \ge k$.

Alimonti [3] has proved that if T is a local optimum for the flip neighborhood and for the objective function f_{NOB}, then T satisfies at least $m/2^k$ clauses. Since obviously $m^* \le m$, we get that the non oblivious local search achieves a performance ratio of 2^k. However by considering the approximation with respect to the value m^* of the optimal solution, it is possible to obtain a better performance ratio, as the following theorem shows.

Theorem 3 (Alimonti, 1995 [4]). *The non-oblivious local search with the flip neighborhood, and the objective function f_{NOB}, is a polynomial time $2^k - 1$-approximation algorithm for* MAX k-CCSP.

Proof. (sketch) Let T be a local optimum with the flip neighborhood and for the objective function f_{NOB}. Recall that S_j is the set of clauses where j literals are false in T. We denote by $|q|_j$ the cardinality of the literal q is in S_j, for $0 \le j \le k$. Then it can be shown that since T is a local optimum,

$$\Delta f_{NOB}(q) = \sum_{j=1}^{k}(L_{j+1} - L_j)|\overline{q}|_j - \sum_{j=0}^{k-1}(L_j - L_{j+1})|q|_j \le 0, \tag{3}$$

for any literal q. After some lengthy calculus and several lemma about the values L_j, Alimonti derives from the inequality 3 that T satisfies at least $m^*/(2^k - 1)$ clauses. □

Theorem 3 can be improved for the special case $k = 2$. Given a solution $s = (s_1, \dots, s_n)$, the extended d-bounded neighborhood $\mathcal{N}_{ed}$ is defined in the following way: $s' \in \mathcal{N}_{ed}(s) \iff (d_H(s, s') \le d) \vee s' = (\overline{s_1}, \overline{s_2}, \dots, \overline{s_n})$. In other words either we flip the values of up to d variables, either we flip all of them. The following theorem uses the non-oblivious objective function $f'_{NOB} = 4|S_0| + |S_1|$.

Theorem 4 (Alimonti, 1997 [5]). *The non-oblivious local search algorithm with the extended flip neighborhood, and the objective function f'_{NOB}, is a polynomial time 2.5-approximation algorithm for* MAX 2-CCSP, *and this bound is tight.*

Proof. (sketch) Let T be a local optimum for the extended flip neighborhood and for the objective function f'_{NOB}. Since T is a local optimum for the flip neighborhood, no literal q exists such that $\Delta f'_{NOB}(T - q \cup \overline{q}) = -3|q|_0 + 3|\overline{q}|_1 - |q|_1 + |\overline{q}|_2 > 0$, therefore

$$3|q|_0 + |q|_1 \ge 3|\overline{q}|_1 + |\overline{q}|_2. \tag{4}$$

Since T is a local optimum for the extended flip neighborhood,

$$|S_0| \ge |S_2|. \tag{5}$$

Alimonti uses inequalities 4 and 5 to obtain that T satisfies at least $m^*/2.5$ clauses. An instance is given in [5] showing that this bound is tight. □

Since the MAX DICUT problem (see Section 8) is a special case of MAX 2-CCSP one obtains.

Corollary 1. *The non-oblivious local search algorithm with the extended flip neighborhood is a polynomial time* 2.5*-approximation algorithm for* MAX DICUT.

3.5 Oblivious Local Search for MAX CSP(k, r)

We consider now the MAX CSP(k, r) problem. In the flip neighborhood two assignments f and f' are neighbors if the number of variables whose values are different in f and f' is exactly one.

Theorem 5 (Halldórsson, Lau, 1997[53]). *The local search algorithm with the flip neighborhood is a polynomial time (k/r)-approximation algorithm for* MAX CSP(k, r). *This bound is tight.*

Proof. (sketch) Since f is a local optimum with respect to the flip neighborhood, it can be shown that for all variables $v \in V$ the number of constraints which are adjacent to v, and that are satisfied by f is at least $\lceil rd(v)/k \rceil$. Then summing up over all vertices, one obtains the result. □

Halldórsson and Lau in [53] also develop a more advanced version of local search. They are not able to improve the k/r ratio for MAX CSP(k, r), but applied to MAX k-CUT they obtain a ratio of $\frac{k}{k-1}\frac{2\Delta+k-1}{2\Delta+k}$. The algorithm works as follows. First an initial assignment is found using a greedy algorithm. Then, several iterations of a modified local search are performed, and finally the local search with the flip neighborhood is performed. Given an assignment f the idea of the modified local search is to consider vertices v such that the number of constraints incident to v, in the constraint (multi) graph, that are not satisfied by f is above a certain level, and to change the value of $f(v)$.

3.6 Applications for Graph and Hypergraph Coloring Problems

We begin by some basic definitions. For more details the reader can consult the book of Berge [16].

An undirected hypergraph (simply called hypergraph) $\mathcal{H}$ is a pair $\langle V, H \rangle$, where V is the set of vertices, H is the set of hyperarcs, and for each $X \in H$, $X \subseteq V$. A coloring is a function that assigns to each vertex a color. A subset of hyperarcs $H' \subseteq H$ is said to be h-colorable, if there exists a coloring such that each hyperarc $X' \in H'$ contains at least one vertex for each one of the h colors. This is the standard definition given in the book of Garey and Johnson [45]. Alimonti [4] also consider another more restrictive version of colorability. A subset of hyperarcs $H' \subseteq H$ is said to be 2-perfect-colorable, if there exists a coloring such that each hyperarc $X' \in H'$ contains, either the same number of vertices (if $|X'|$ is even), or the same number of vertices but one (if $|X'|$ is odd), for each one of the 2 colors. The hypergraph h-colorability (resp. 2-perfect-colorability) problem is the problem of finding a h-colorable (resp. 2-perfect-colorable) subset

$H' \subseteq H$ of maximum cardinality $|H'|$. The hypergraph 2-colorability problem is also known as the SET SPLITTING problem [45].

A directed hypergraph $\mathcal{H}$ is a pair $\langle V, H\rangle$, where V is the set of vertices, H is the set of directed hyperarcs, and for each $\langle X, v\rangle \in H$, $X \subseteq V$ and $v \in V$. The directed hypergraph 2-colorability problem is the problem of finding a partition of V in two sets S and $\overline{S}$ that maximize the cardinality of $|H'|$, where $H' \subseteq H$ and for each hyperarc $\langle X, v\rangle \in H'$ one has $X \subseteq S$ and $v \in \overline{S}$. Notice that if $\mathcal{H}$ is a graph, this problem becomes the MAX DICUT problem (see Section 8).

By representing instances of hypergraph colorability problems as instances of the conjunctive constraint satisfaction problem it is possible to get from Theorem 3 approximation ratio results for hypergraph colorability problems. By this way the following four theorems 6, 7, 8 and 9 are obtained.

Let $\mathcal{H}$ be a hypergraph, with $|H| = m$ hyperarcs of size k.

Theorem 6 (Alimonti, 1995[4]). *The hypergraph 2-colorability problem on* $\mathcal{H}$ *can be solved within approximation ratio* $2^{k-1}/(2^{k-1}-1)$.

Theorem 7 (Alimonti, 1995[4]). *The hypergraph 2-perfect-colorability problem on* $\mathcal{H}$ *can be solved within approximation ratio* $\sqrt{\pi k/2}$ *if* k *is even, and* $\frac{1}{2}\sqrt{\pi(k+1)/2}$ *if* k *is odd.*

We consider now that the size of each hyperarc is $k = h$. The hypergraph h-colorability problem on $\mathcal{H}$ means that we look for a coloring for which no nodes on a hyperarc have the same color.

Theorem 8 (Alimonti, 1995[4]). *The hypergraph* h*-colorability problem on* $\mathcal{H}$ *can be solved within approximation ratio* $\frac{e^k}{\sqrt{2\,\pi\,k}}\,(\frac{k+1}{k})^k$.

Alimonti shows that directed hypergraph 2-colorability problem cannot be approximated by means of oblivious local search, even if we restrict to the case of directed graphs. For any approximation ratio r, whatever neighborhood size b is allowed (the number b of vertices which can change their color to obtain a neighboring solution) there are local optima solutions with respect to such neighborhood that do not guarantee the ratio r. In contrast, by representing this problem as a satisfiability problem we get from Theorem 3 the following result.

Theorem 9 (Alimonti, 1995[4]). *Let* $\mathcal{H} = \langle V, H\rangle$ *be a directed hypergraph, with* $|H| = m$ *hyperarcs of size* k*. The directed hypergraph 2-colorability problem on* $\mathcal{H}$ *can be solved within approximation ratio* $2^k - 1$.

When $k = 2$ we have the MAX DICUT problem (see Section 8), and Theorem 9 cannot compete with the ratio obtained by Goemans and Williamson [46].

4 The Traveling Salesman Problem

In this famous problem in combinatorial optimization, we are given a complete graph over n vertices, with nonnegative edge costs. The traveling salesman problem, denoted by TSP, is to find a minimum cost cycle (tour T) visiting every vertex exactly once.

Approximation results:
It is an NP-hard problem and cannot be approximated (see e.g. [100]). However if we assume that the edge costs satisfy the triangle inequality, i.e. we consider the METRIC TSP, then the problem becomes approximable. The well known algorithm of Christofides [32] achieves an approximation ratio of 3/2.

Neighborhoods:
The standard neighborhoods for TSP are the 2-opt and k-opt ($k \geq 3$) neighborhoods. Given a tour T, a tour T' belongs to the k-opt neighborhood of T if T' can be obtained from T by removing at most $k' \leq k$ edges from T and adding k' new edges. An example of 2-opt move is depicted in Figure 1.

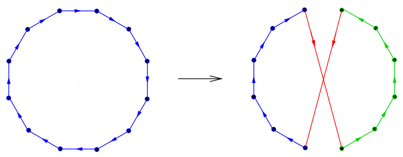

Fig. 1. An illustration of the 2-opt move

The well known Lin-Kernighan algorithm, which gives in practice very good solutions, is based on such neighborhoods [80]. Very roughly, in this algorithm the value of k is allowed to vary during the execution of the algorithm and when an improvement is found the search continues for finding an even better move.

Local search complexity:
The traveling salesman problem is PLS-complete under the k-opt move for some constant k [75]. It is also PLS-complete under the LK' neighborhood [87] (a variant of the Lin-Kernighan neighborhood, see [1]).
Lueker [81] constructed an instance for which there exists an exponentially long sequence of improving 2-opt moves. Chandra, Karloff, Tovey [28] have extended Lueker's construction for all $k > 2$, giving explicit instances for which there exist exponentially long improving sequences of k-opt moves. For all sufficiently large k, Johnson, Papadimitriou and Yannakakis [66] and Krentel [75] have proved the existence of instances and initial tours such that *any* local search using k-opt and starting from this initial tours needs an exponential number of steps before converging to a local optimal solution. Krentel claimed to have extended this result to all $k \geq 8$. This result also holds for the LK' neighborhood [87]. However it is not known whether this results hold for the 2-opt and 3-opt neighborhoods, or if there always exist a way of choosing an improvement such that the local search algorithm finds a local optimum solution after a polynomial number of moves.

Local search approximation results:
Chandra, Karloff, Tovey [28] have obtained several approximation and convergence results concerning the performance of local search for various instances of the traveling salesman problem. We cite here only a few of them. For random TSP instances in the unit square they show that the expected number of iterations before reaching a local optimum is polynomial, and the expected worst-case approximation ratio is bounded by a constant. For METRIC TSP they show that a local search using the 2-opt neighborhood achieves a performance ratio of at most $4\sqrt{n}$ for all n. Moreover they show that using a larger neighborhood does not fundamentally improve the ratio since a local search with the k-opt neighborhood can have a performance ratio at least $\frac{1}{4}n^{1/2k}$ for infinitely many n.

Theorem 10 (Chandra, Karloff, Tovey, 1999[28]). *A local search algorithm with the 2-opt neighborhood achieves a $4\sqrt{n}$ approximation ratio for* METRIC TSP.

Proof. (sketch) We denote by $wt(e)$ the weight of edge e. Let C_{opt} denote the cost of an optimal tour. and let T be any tour of cost C, which is locally optimal with respect to the 2-opt neighborhood. Let E_k, for $k \in \{1, \ldots, n\}$ the set of big edges of T: $E_k = \{e \in T \mid wt(e) > 2C_{opt}/\sqrt{k}$. Then the first part of the proof is to show that $|E_k| < k$. Assuming this last result is true, then it means that the weight of the k-th largest edge in T is at most $2C_{opt}/\sqrt{k}$, therefore

$$\begin{aligned} C &= \sum_{k=1}^{n} wt(k - \text{th largest edge}) \\ &\leq 2C_{opt} \sum_{k=1}^{n} \frac{1}{\sqrt{k}} \\ &\leq 2C_{opt} \int_{x=0}^{n} \frac{1}{\sqrt{x}} dx \\ &= 4\sqrt{n} C_{opt}. \end{aligned}$$

The proof of $|E_k| < k$ is by contradiction. Here we give only an idea of the proof. Let V be the set of vertices of the instance. For any subset of vertices $V' \subseteq V$ we denote by $C_{opt}(V')$ the length of a shortest tour on V'. We give an arbitrary orientation of the tour T. Let $t_1, \ldots, t_r$, with $r = |E_k| \geq k$ be the tails of each arc from E_k in tour T. Then it can be shown that there exists at least $\sqrt{k}$ tails which are at a distance at least $C_{opt}/\sqrt{k}$ from each other. Consider the traveling salesman instance restricted on this set V' of tails. Then the shortest tour on this set has a length $C_{opt}(V')$ greater than $\sqrt{k}\, C_{opt}/\sqrt{k} = C_{opt}$ contradicting the fact that since the distances satisfy the triangular inequality then for any subset $V' \subseteq V$ one has $C_{opt}(V') \leq C_{opt}(V)$. □

We consider now the restricted version of the traveling salesman problem, denoted by TSP(1,2), where each edge has a cost of 1 or 2. This problem remains NP-hard and Papadimitriou and Yannakakis [88] proposed a polynomial time

7/6-approximation algorithm for this problem. Their algorithm uses a subtour patching technique.

Theorem 11 (Khanna, Motwani, Sudan and Vazirani, 1994 [70,71]). *A local search algorithm with the 2-opt neighborhood achieves a* $3/2$ *approximation ratio for* TSP(1,2)*, and this bound is asymptotically tight.*

Proof. Let $C = v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n} v_{\pi_1}$ be a local optimum solution with the 2-opt neighborhood. Let O be any optimal solution. To each unit cost edge e in O we associate a unit cost edge e' in C as follows. Let $e = (v_{\pi_i}, v_{\pi_j})$ with $i < j$. If $j = i + 1$ then $e' = e$. Otherwise e' is a unit cost edge among $e_1 = (v_{\pi_i}, v_{\pi_{i+1}})$ and $e_2 = (v_{\pi_j}, v_{\pi_{j+1}})$. Indeed, either e_1 or e_2 must be of unit cost. If it is not the case, then the tour C' obtained from C by removing edges e_1 and e_2 and adding edges e and $f = (v_{\pi_{i+1}}, v_{\pi_{j+1}})$ has a cost at least one less than C and therefore C would not be a local optimal solution with the 2-opt neighborhood.

Let U_O denotes the set of unit cost edges in O, and U_C the set of unit cost edges in C obtained from U_O using the above mapping. Since an edge $e' = (v_{\pi_i}, v_{\pi_{i+1}})$ in U_C can only be the image of unit cost edges incident on v_{π_i} in O and since O is a tour, there are at most two edges in U_O which map to e'. Thus $|U_C| \geq |U_O|/2$ and we obtain $\frac{cost(C)}{cost(O)} \leq \frac{|U_O|/2+2(n-|U_O|/2)}{|U_O|+2(n-|U_O|)} \leq \frac{3}{2}$.

The above bound can be shown to be asymptotically tight [70]. □

5 The Quadratic Assignment Problem

Given two $n \times n$ symmetric matrices $F = (f_{ij})$ and $D = (d_{ij})$, with a null diagonal, the (symmetric) Quadratic Assignment Problem, denoted by QAP, can be stated as follows:

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{k=i+1}^{n} f_{ik} d_{\pi(i)\pi(k)},$$

where Π is the set of all permutations of $\{1, 2, \ldots, n\}$. One of the major applications of the QAP is in location theory where f_{ij} is the flow of materials from facility i to facility j, and d_{ij} represents the distance from location i to location j [73]. The objective is to find an assignment of all facilities to locations which minimizes the total cost (see [89] for a survey of this problem).

This problem is a generalization of the traveling salesman problem and is therefore NP-hard. Due to time constraints, branch and bound methods to solve the QAP exactly remain limited to instances of small size. On December 2000 Peter Hahn et al. [24] solved Kra30b, an instance with $n = 30$. The computation time needed amounted to 182 days on a single cpu HP-3000 workstation. However local search based algorithms are usually very good at providing approximate solutions.

Neighborhoods:
We consider the 2-exchange neighborhood which is the standard one for the QAP. Given a permutation $\pi = (\pi(1), \ldots, \pi(i), \ldots, \pi(j), \ldots, \pi(n))$, its neighbors

are the $\frac{n(n-1)}{2}$ permutations of the form $(\pi(1), \ldots, \pi(j), \ldots, \pi(i), \ldots, \pi(n))$ for $1 \leq i < j \leq n$, obtained from π by performing a swap.

Local search complexity:
In [94], it has been proved that the Graph Bipartitioning Problem is PLS-complete for the swap neighborhood, and since it is a particular case of the symmetric QAP with the 2-exchange neighborhood, it follows that the QAP with this neighborhood is PLS-complete.

If we denote by C_{loc} the cost of any solution which is a local minimum relative to a certain neighborhood and C_{AV} the average cost over all possible solutions, it has been proved for the Traveling Salesman Problem that $C_{\text{loc}} \leq C_{\text{AV}}$ with the 2-exchange neighborhood [49], the 3-exchange and the 2-opt neighborhood [33].

Let $s(A)$ denote the sum of all terms of a given matrix A. Let x and y two vectors of the same dimension. The maximum (respectively minimum) scalar product of x and y is defined by: $\langle x, y\rangle_+ = \max_{\pi \in \Pi} \langle x, \pi y\rangle$ (respectively $\langle x, y\rangle_- = \min_{\pi \in \Pi} \langle x, \pi y\rangle$). Let F_k and D_k denote the sum over the k-th column of F and D respectively, and let $\langle F, D\rangle_+$ (respectively $\langle F, D\rangle_-$) be an abbreviation for $\langle (F_1, \ldots, F_n), (D_1, \ldots, D_n)\rangle_+$ (respectively $\langle (F_1, \ldots, F_n), (D_1, \ldots, D_n)\rangle_-$).

Theorem 12 (Angel, Zissimopoulos, 1998[7]). *For the* QAP, *let* C_{loc} *the cost of any solution found by a deepest local search with the 2-exchange neighborhood, then the following inequality holds:* $C_{\text{loc}} \leq \frac{\langle F, D\rangle_+}{s(F)s(D)}\, n\, C_{\text{AV}}$, *where* C_{AV} *is the average cost of all possible permutations.*

Proof. The demonstration's technique is inspired from [49]. Let $\pi = (\pi(1), \ldots, \pi(n))$ any permutation, and π_{ij} the permutation obtained from π by swapping $\pi(i)$ and $\pi(j)$. Let $C(\pi)$ the cost of the permutation π: $C(\pi) = \frac{1}{2}\sum_{i,j=1}^{n} f_{ij} d_{\pi(i)\pi(j)}$. We define $\delta_{ij} = C(\pi_{ij}) - C(\pi)$. Then, after some calculus one easily obtains that the total difference cost relative to the 2-exchange neighborhood is given by

$$\sum_{i,j=1}^{n} \delta_{ij} = 2s(F \cdot \pi(D)) - 4(n-1)C(\pi), \tag{6}$$

with the matrix $\pi(D)$ defined by $(\pi(D))_{ij} = d_{\pi(i)\pi(j)}$.

In the same way, it can be shown that the average cost over all permutations is given by

$$C_{\text{AV}} = \frac{s(F)s(D)}{2n(n-1)}. \tag{7}$$

We note, by analogy with the space Laplacian operator, $\nabla^2 C(\pi) = \frac{\sum_{i,j=1}^{n} \delta_{ij}}{n(n-1)}$ the average cost difference between the permutation π and its neighbors. Using equality 6 one has $\nabla^2 C(\pi) = \frac{-4C(\pi)}{n} + \frac{2s(F \cdot \pi(D))}{n(n-1)}$.

The key idea is to bound $\nabla^2 C(\pi)$ and to use the fact that at each step there always exists an exchange i, j such that $\delta_{i,j} \leq \nabla^2 C(\pi)$. By using equality 7 we obtain $\nabla^2 C(\pi) = \frac{-4\overline{C}(\pi)}{n} + \alpha(\pi)$, where $\overline{C}(\pi) = C(\pi) - C_{\mathrm{AV}}$ and $\alpha(\pi) = \frac{2s(F\cdot\pi(D))}{n(n-1)} - \frac{2s(F)s(D)}{n^2(n-1)}$. Let $\alpha = \max_\pi \alpha(\pi) = \max_\pi \frac{2}{n(n-1)}(s(F\cdot\pi(D)) - \frac{s(F)s(D)}{n})$. Then, $\nabla^2 C$ is bounded above by $\frac{-4}{n}(C(\pi) - C_{\mathrm{AV}}) + \alpha$.

So, since there always exists an exchange i, j such that $\delta_{i,j} \leq \nabla^2 C(\pi)$, the deepest local search algorithm picks a movement at least as good as this one, and the cost of the solution is at the next step less than $C(\pi) - \frac{4}{n}(C(\pi) - C_{\mathrm{AV}}) + \alpha$. Iterating this process gives a limit cost l which verifies $l = l - \frac{4}{n}(l - C_{\mathrm{AV}}) + \alpha$. It follows that $l = \frac{\alpha n}{4} + C_{\mathrm{AV}}$. By using equality 7 and some calculus it can be shown that

$$\frac{l}{C_{\mathrm{AV}}} = \max_\pi \frac{s(F\cdot\pi(D))}{s(F)s(D)}\, n \leq \frac{\langle F, D\rangle_+}{s(F)s(D)}\, n \leq \frac{1}{2} n\,. \qquad \square$$

It would be interesting to have an upper bound for C_{loc} where the size n of the problem doesn't occur. But notice that if we conserve the shape of the bound of theorem 12, that is the term $\frac{\langle F,D\rangle_+}{s(F)s(D)}$, the bound cannot be improved. Indeed, recall that when restricted to the Symmetric Traveling Salesman Problem, this bound becomes C_{AV}, a result which is optimal (think at the special case when all entries of the distance matrix are equal, then all the solutions have the same cost equal to C_{AV}).

Corollary 2. *For the* QAP*, the following inequality holds: $C_{\mathrm{loc}} \leq \frac{n}{2} C_{\mathrm{AV}}$. Moreover, there is a sequence of instances for which the ratio $C_{\max}/\frac{n}{2}C_{\mathrm{AV}}$ tends to infinity, where $C_{\max}$ is the maximum cost over all permutations.*

Theorem 13 (Angel, Zissimopoulos, 1998[7]). *If the matrices F and D are positive integer ones, a deepest descent local search will reach a solution with a cost less than $\frac{n}{2}C_{\mathrm{AV}}$ in at most $\mathcal{O}(n \log(s(F)s(D)/2))$ iterations.*

Proof. We have seen that at each step the cost $C(\pi)$ of the current solution π becomes less than $C(\pi) - \frac{4}{n}(C(\pi) - C_{\mathrm{AV}}) + \alpha$. If we note $\tilde{C}(\pi) = C(\pi) - l$ (l is the limit cost defined above) it is equivalent to say that a each step, $\tilde{C}(\pi)$ becomes less than $(1 - \frac{4}{n})\tilde{C}(\pi)$. Indeed, at each step $C(\pi) - l$ becomes less than $C(\pi) - \frac{4}{n}(C(\pi) - C_{\mathrm{AV}}) + \alpha - (\frac{\alpha n}{4} + C_{\mathrm{AV}}) = (1 - \frac{4}{n})(C(\pi) - \frac{\alpha n}{4} - C_{\mathrm{AV}})$. Since F and D are integer matrices, the cost at each step must decrease at least one. So, if we have L such that $C_{\max} - l \leq 2^L$ the number of steps t until $\tilde{C}(\pi)$ is less than zero satisfies $(1 - \frac{4}{n})^t < 2^{-L}$, that is $t > \frac{-L}{\log_2(1-\frac{4}{n})} \sim \mathcal{O}(nL)$ when $n \to \infty$. We have $C_{\max} - l \leq C_{\max} \leq \frac{s(F)s(D)}{2}$, and the result follows. $\square$

Notice that, we do not claim that the entire local search will use at most this polynomial number of iterations. Since the QAP is PLS-complete, this would be equivalent to say that $P = NP$! Certainly, for some instances the search will go on.

Theorem 12 can be applied to some particular cases of QAP. The symmetric traveling salesman problem with n cities can be seen as a particular case of QAP by considering D to be the distance matrix and F to be defined by $f_{i,i+1} =$

$f_{i+1,i} = 1$ with $1 \leq i \leq n-1$, $f_{n,1} = f_{1,n} = 1$ and $f_{ij} = 0$ otherwise. Using Theorem 12 we obtain for the 2-exchange neighborhood (any pair of cities can be exchanged in this neighborhood), $C_{\mathrm{loc}} \leq C_{\mathrm{AV}}$, a result also previously found by Grover [49].

Recall that given a graph with an even number n of vertices the graph bipartitioning problem consists of finding a partition of the vertices into two equal-sized subsets A and B such that the number of edges having one extremity in A and the other in B is minimized. For this problem D is the adjacency matrix of the graph, and

$$F = \begin{pmatrix} 0 & U \\ U & 0 \end{pmatrix},$$

where U is the $\frac{n}{2} \times \frac{n}{2}$ matrix, with $u_{ij} = 1$, $i, j = 1, \ldots, \frac{n}{2}$. Using Theorem 12, we obtain for the swap neighborhood (any vertex belonging to A can be swapped with a vertex belonging to B in this neighborhood) $C^{-}_{\mathrm{loc}} \leq C_{\mathrm{AV}}$, a result previously found by Grover [49].

Others applications concerning generalized maximum independent set and maximum clique problems are considered in [7].

6 Set Packing and Independent Set

Given a collection of weighted sets, each containing at most k elements from a finite base set, the weighted k-set packing problem, denoted by W-k-SET PACKING, is to find a maximum weight subcollection of disjoint sets. In the unweighted case, denoted by k-SET PACKING, all sets have a weight of one, and the problem is to find a maximum cardinality subcollection of disjoint sets. When $k = 2$ there is a polynomial reduction to the maximum weighted matching problem in graphs and the problem is therefore polynomially solvable. However the problem becomes NP-hard for any $k \geq 3$, even in the unweighted case [45].

The simplest heuristic for W-k-SET PACKING is the greedy one: start from the empty subcollection, then add to the current subcollection a set of maximum weight from the base set, then remove it and all sets that intersect it in the base set, and repeat until the base set becomes empty. It is easy to see that this greedy algorithm achieves a performance ratio of k and this bound is tight.

In local search at each iteration we attempt to replace some subsets of the current solution by some collection of subsets of greater total weight that does not intersect the remainder of the solution. In order to obtain polynomial time algorithms there is a restriction on the number of sets added. For a fixed t, at each iteration we check whether there are $p \leq t$ disjoint sets that are not in the current solution that intersect at most $p-1$ sets that are in it. If this is the case, we swap the sets to form a larger collection. A solution is said to be t-optimal if such a swap does not exist.

For k-SET PACKING Hurkens and Schrijver [63] showed that a local search algorithm leads to an approximation ratio of $k/2 + \epsilon$, where ϵ depends on t, i.e. the size of the neighborhood used. Halldórsson [52] proposed a restricted form of this local search with the same performance ratio but a decreased complexity.

For W-k-SET PACKING, Arkin and Hassin [8,9] showed that a local search algorithm yields an approximation ratio $k-1+\epsilon$, this bound being tight. Independently Bafna, Narayanan and Ravi [13] obtained an approximation ratio of $k-1+\frac{1}{k}$. Chandra and Halldórsson [27] combining the greedy and local search approach, starting from an initial solution found by the greedy algorithm and then performing a deepest local search, obtained a performance ratio of $2(k+1)/3$, and showed it was asymptotically tight. As we will see in Section 6.3, Berman [17] using a non-oblivious local search obtained a $(k+1)/2$ approximation ratio. Instead of considering the sum of weights of the sets in the solution he considers the sum of squares of the weights of the sets.

The W-k-SET PACKING problem is a special case of the weighted independent set problem. Given an undirected graph $G(V,E)$ the maximum independent set problem, denoted by INDEPENDENT SET, ask to find a set of nodes $A \subseteq V$ of maximum cardinality such that there is no edge in G between any couple of nodes from A: $\forall v_i, v_j \in A$, $[v_i, v_j] \notin E$. In the weighted version, denoted by W-INDEPENDENT SET, there is a weight $w(v) \in \mathbb{N}$ associated to each vertex $v \in V$, and one seek for a maximum weighted $w(A) = \sum_{v\in A} w(v)$ set of non adjacent vertices A. Given an instance of the weighted k-set packing problem, we can build an intersection graph where each vertex corresponds to a set, and there is an edge between two vertices S_i and S_j if and only if $S_i \cap S_j \neq \emptyset$. Finding a maximum weighted k-set packing reduces to the problem of finding a maximum weighted independent set in the intersection graph. Notice now that since each set is of cardinality at most k, the intersection graph is $(k+1)$-claw free, i.e. it has the property that no induced subgraph contains a $k+1$-claw which is a set of $k+1$ independent vertices (i.e. mutually nonadjacent) that share a common neighbor. However the class of $(k+1)$-claw free graphs properly includes these intersection graphs.

Some of the above mentioned results remains valid for the weighted and unweighted independent set problem on $(k+1)$-claw free graphs. This time in local search at each iteration we attempt to replace a subset of vertices by another subset of vertices of greater total weight that are not adjacent with the remainder of the solution.

6.1 Oblivious Local Search for k-SET PACKING

Theorem 14 (Hurkens, Schrijver, 1989[63]). *Let $m, n \in \mathbb{N}$, let V be a set of size n, and let $E_1, \dots, E_m$ be subsets of V. Furthermore, let $k, t \in \mathbb{N}$ with $k \geq 3$ such that the following holds:*

(i) Each element of V is contained in at most k of the sets $E_1, \dots, E_m$.
(ii) For any $p \leq t$, any p of the sets among $E_1, \dots, E_m$ cover at least p elements of V.

Then

$$\frac{m}{n} \leq \begin{cases} \frac{k(k-1)^r - k}{2(k-1)^r - k} & if \quad t = 2r-1; \\ \frac{k(k-1)^r - 2}{2(k-1)^r - 2} & if \quad t = 2r. \end{cases}$$

Moreover these bounds are tight.

If we regard V and $H = \{E_1, \ldots, E_m\}$ as a hypergraph (V, H). It becomes clear that under the hypothesis of (i) and (ii) there is some bound on the ratio m/n. The proof involves a quite involved induction.

This result can be used to obtain an approximation ratio for local search applied to k-SET PACKING as follows. Let $X_1, \ldots X_q$ be the collection of sets, each containing at most k elements. Let V be the index set of the sets in some t-optimal solution, and let O be the index set of the sets in some fixed optimal solution. Let E_i be the index set of the sets in the t-optimal solution which intersect X_i, for $i \in O$. Since each set X_i for $i \in V$ has a size at most k, the condition (i) of the theorem is satisfied. Moreover from the definition of a t-optimal solution we get that the condition (ii) of the theorem is also satisfied. We have $|V| = n$ and $|O| = m$, therefore the right-hand side of the inequality of Theorem 14 bounds the error ratio $|O|/|V|$.

6.2 Oblivious Local Search for W-INDEPENDENT SET on d-Claw Free Graphs

Recall that in an undirected graph, a d-claw C is an induced subgraph that consists of a node Z_C called the *center* of the claw and an independent set T_C of d nodes, called *talons* of the claw, each one connected to the center Z_C. In the following we consider only d-claw free graphs, i.e. graphs that possess no d-claws. Recall that an approximation result for W-INDEPENDENT SET on d-claw free graphs, when $d = k + 1$, immediately leads to an approximation result for W-k-SET PACKING.

The simplest algorithm for W-INDEPENDENT SET is certainly the greedy algorithm. Let us consider an undirected graph $G(V, E)$. For $K, L \subset V$ let $N(K, L) = \{u \in L \ : \ \exists v \in K \text{ such that } [u, v] \in E \text{ or } u = v\}$. $N(K, L)$ is the set of nodes in L which are adjacent to at least one node from K, union the set of nodes both in K and L. The greedy algorithm 2 starts from an empty set A, and while $V \setminus N(A, V)$ is non-empty, it choose a node of maximum weight from it and add it to the set A. It can be shown that this algorithm achieves a performance ratio $d - 1$.

The greedy algorithm can be considered as a deepest descent local search algorithm by considering that the neighborhood of the solution $A \subset V$ (the current independent set) is defined by $\mathcal{N}(A) = \bigcup_{v \in V \setminus N(A,V)} A \cup \{v\}$. A natural question is whether this approximation ratio can be improved by enlarging the size of the neighborhood. It is indeed the case.

Algorithm 2 GREEDY for W-INDEPENDENT SET

```
$A \leftarrow \emptyset$
while $V \setminus N(A, V) \neq \emptyset$ do
    choose a node $u \in V \setminus N(A, V)$ of maximum weight $w(u)$
    $A \leftarrow A \cup \{u\}$
end while
return $A$
```

Algorithm 3 SizeTwoImp for independent set (from Bafna, Narayanan and Ravi [13])

```
$A \leftarrow \emptyset$
while there exists $\{u, v\}$ that improves $|A|$ do
  $A \leftarrow A \setminus N(\{u, v\}, A) \cup \{u, v\}$
end while
return $A$
```

A node set C *improves* the independent set A if $w((A \setminus N(C, A)) \cup C) > w(A)$. The set C is also called an improvement. The payoff factor of C is defined by $w(C)/w(N(C, A))$. For $\alpha > 1$ an α-good improvement is an improvement with payoff factor at least α. A solution is α-locally optimal if it has no β-good improvement, for any $\beta > \alpha$.

The previous greedy algorithm considered only node sets of size one. The algorithm 3 considers node sets of size two and has a better performance ratio $d/2$ [13]. By increasing the size of allowed improvements, one can obtain polynomial time algorithms for independent set with a performance ratio approaching $(d-1)/2$ [63]. Chandra and Halldórsson have extended this approach for w-independent set [26,27]. The algorithm 4 has an approximation ratio $\frac{2}{3}(d+1)$ which is asymptotically tight.

In order to prove this result, Chandra and Halldórsson introduce the concept of a *projection* of an optimal solution OPT to a given maximal solution I. The representative of each vertex $b \in OPT$ is the vertex of maximum weight in $N(b, I)$. Then the projection of OPT to I is defined as $proj(I) = \sum_{b \in OPT} w(f_I(b))$. The projection has the following three properties. The first one is $proj(Gr) \geq C_{OPT}$, which means that the projection of the greedy solution Gr is initially as large as the optimal solution value C_{OPT} of solution OPT. The second one is: If I' is obtained from I by one or more α-good improvements, then $\frac{k}{\alpha-1}w(I') + proj(I') \geq \frac{k}{\alpha-1}w(I) + proj(I)$, meaning that the more the value of the projection decreases and the more the value of the solution I increases. The last one is: For an α-locally optimal solution I one has $(k+1)w(I) \geq \frac{1}{\alpha}C_{OPT} + proj(I)$, meaning that the weight of a locally optimal solution is large compared with the final projection. So either the value of the projection decreases a lot during the execution of the local search algorithm and by the second property the value of the solution I improves a lot, either the value of the projection is important and by the last property it means that the weight of a locally optimal solution is large with respect to the projection, and by the first property we can conclude that it is large with respect to the optimal solution.

The local search algorithm is not polynomial since the length of the improvement sequence can be exponential in the length of the input instance. In order to get a polynomial time algorithm Chandra and Halldórsson (see also [17]) show how to modify algorithm 4. The idea is to start the greedy algorithm to obtain an initial solution A, then the local search algorithm 4 is runned with a scaled weight function w such that $w(A) = l|V|$, with $l > 1$ an integer. The initial solution is not too far away from the optimal solution since the greedy algorithm has an approximation ratio $d-1$, and the scaling step ensures a polynomial number of steps until reaching a local optimum solution using the usual argument that at each step the weight $w(A)$ must increase by at least one. The approximation ratio, for this polynomial time algorithm, becomes $\frac{l}{l-1}\,\frac{2}{3}\,d$ and the running time is polynomial in ln.

Algorithm 4 BESTIMP for W-INDEPENDENT SET (from Chandra and Halldórsson [26,27])

```
$A \leftarrow \emptyset$
while there exists claw $C$ such that $T_C$ improves $w(A)$ do
  if $V \setminus N(A,V) \neq \emptyset$ then
    choose $u \in V \setminus N(A,V)$ with maximum $w(u)$
    $C \leftarrow \{u\}$
  else
    choose claw $C$ that maximizes $\frac{w(T_C)}{w(N(T_C,A))}$
  end if
  $A \leftarrow A \setminus N(T_C,A) \cup T_C$
end while
return $A$
```

6.3 Non-oblivious Local Search for W-INDEPENDENT SET on *d*-Claw Free Graphs

Berman has obtained a stronger performance ratio for W-INDEPENDENT SET using a non oblivious local search [17]. Let $w^2(A) = \sum_{v \in A} w(A)^2$.

Algorithm 5 SQUAREIMP for W-INDEPENDENT SET (from Berman [17])

```
$A \leftarrow \emptyset$
while there exists claw $C$ such that $T_C$ improves $w^2(A)$ do
  $A \leftarrow A \setminus N(T_C,A) \cup T_C$
end while
return $A$
```

For any node u, let $n(u)$ be a node $v \in N(u,A)$ with the maximum value of $w(v)$. Let

$$charge(u,v) = \begin{cases} w(u) - w(N(u,A))/2 \text{ if } v = n(u) \\ 0 \text{ otherwise} \end{cases}$$

Let C be a claw with center $Z_C = \{v\}$. Then C is said to be a *good* claw if either $N(T_C,A) = \emptyset$ or ($v \in A$ and $\sum_{u \in T_C} charge(u,v) > w(v)/2$). A *nice* claw is a good claw that is minimal in size.

Lemma 1 (Berman, 2000 [17]). *The approximation ratio of* SQUAREIMP *algorithm is $d/2$, and this ratio is tight.*

Proof. (sketch) It can be proved that if C is a nice claw, then T_C improves $w^2(A)$ (but not necessarily $w(A)$). Therefore, when SQUAREIMP terminates we obtain a solution A such that no good claw exists. Let A^* be a maximum weighted independent set. If we can distribute the weight $w(A^*)$ among all the nodes of A in such a way that no node $v \in A$ receives more than $dw(v)/2$, then it shows that $w(A^*)/w(A) \leq d/2$. In a first step each $u \in A^*$ sends to each $v \in N(u, A)$ a weight equal to $w(v)/2$. Thus u sends a weight of $w(N(u, A))/2$. In the second step, u sends its remaining weight $w(u) - w(N(u, A))/2$, that is $charge(u, n(u))$, to $n(u)$. On the receiving side, in the first step a node $v \in A$ gets at most $(d-1)w(v)/2$ from all its neighbors in A^* (notice that there are at most $d-1$ of them otherwise they would form talons of a d-claw). In the second step, v gets at most $w(v)/2$ otherwise the nodes that send weights to v would form talons of a good claw.

Figure 2 depicts an example showing that for 4-claw free graphs the approximation ratio is at least 2. Algorithm SQUAREIMP may start by picking, one by one, elements of set Y. Such examples can be constructed for general d-claw free graphs. □

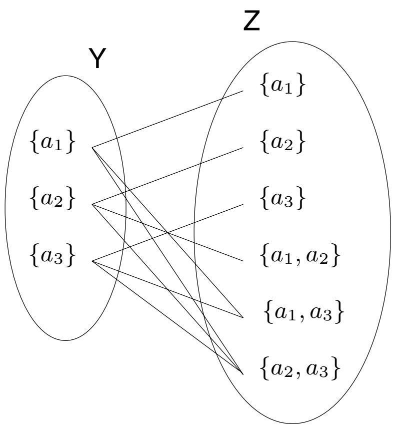

Fig. 2. The approximation ratio of SQUAREIMP is at least $d/2$ on d-claw free graph. Here $d = 4$ and the 9 vertices have a weight equal to one.

Theorem 15 (Berman, 2000 [17]). *For every d there exists a non-oblivious local search algorithm that given a d-claw free graph with n nodes and $l > 1$, finds a solution to with approximation ratio $\frac{l}{l-1}\frac{d}{2}$ in time that is polynomial in ln.*

6.4 INDEPENDENT SET on Bounded Degree Graphs

We assume that the degree of any vertex is bounded by a constant Δ. The neighborhood of an independent set S is the set of all independent sets S' which can be obtained from S by either, adding to S a single vertex not in S, or removing a vertex from S and adding two vertices not in S. We call this neighborhood 3-flip.

Lemma 2 (Khanna, Motwani, Sudan and Vazirani, 1994[70,71]). *The local search algorithm with the 3-flip neighborhood has a performance ratio $(\Delta + 1)/2$.*

Proof. Let I be a local optimum solution found by the local search algorithm, and let O be any optimal independent set. We let $X = I \cap O$.

Since we cannot add a vertex outside I to the set I to obtain a larger independent set, it means that each vertex in $V \setminus I$ must have at least one incoming edge to I. Since we cannot remove one vertex from I and add two vertices outside I to get a larger independent set it means that at most $|I \setminus X|$ vertices in $O \setminus I$ can have exactly one edge coming into I. Thus $|O \setminus X| - |I \setminus X| = |O| - |I|$ vertices in $O \setminus X$ must have at least two edges entering I.

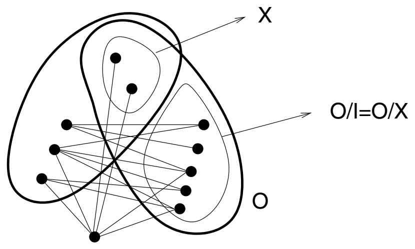

Fig. 3. An illustration of the proof of Lemma 2

Therefore the minimum number of edges between I and $O \setminus X$ is $|I \setminus X| + 2(|O| - |I|)$. On the other hand, the maximum number of edges between I and $O \setminus X$ is bounded by $\Delta|I \setminus X|$. Thus $\Delta|I \setminus X| \geq |I \setminus X| + 2(|O| - |I|)$, using $|I \setminus X| = |I| - |X|$ we get $(\Delta + 1)|I| \geq (\Delta - 1)|X| + 2|O|$ and so $\frac{|I|}{|O|} \geq \frac{\Delta - 1}{\Delta + 1}\frac{|X|}{|O|} + \frac{2}{\Delta + 1}$. □

Halldorsson and Radhakrishnan [54] have shown that the local search algorithm with the 3-flip neighborhood when run on k-clique free graphs yields an independent set of size at least $2n/(\Delta + k)$. They combine this local search algorithm with a clique-removal scheme to achieve a performance ratio of $\Delta/6(1 + o(1))$.

Theorem 16 (Khanna, Motwani, Sudan and Vazirani, 1994[70,71]). *An approximation algorithm which simply outputs the larger of the two independent sets computed by the local search algorithm and the classical greedy algorithm has performance ratio $(\sqrt{8\Delta^2 + 4\Delta + 1} - 2\Delta + 1)/2$.*

The performance ratio given in Theorem 16 is essentially $\Delta/2.414$. This improves upon the approximation ratio of $\Delta/2$ due to Hochbaum [59], when $\Delta \geq 10$. Berman and Fürer [18] gives an algorithm with performance ratio $(\Delta + 3)/5 + \epsilon$ when Δ is even, and $(\Delta + 3.25)/5 + \epsilon$ when Δ is odd.

7 The Set Cover Problem

In the set covering problem we are given a base set U, a collection C of subsets of U whose union is U, and the objective is to find a minimal size sub-collection of C whose union is still U. We will assume that each subset in C has size at most k, with k a constant. In this case the problem is called k-SET COVER. Without loss of generality, we can assume that the collection of subsets is closed under subsets, i.e. for any subset all its subsets belong to C. In the following a set of size k is called a k-set. For $k = 2$, this problem can be solved in polynomial time by matching techniques. For $k \geq 3$, the k-SET COVER problem is NP-hard [45] and is MAX SNP-hard [70].

The greedy algorithm chooses a set that covers the maximum number of uncovered elements and updates the remaining sets. One can shows that the performance ratio of the greedy algorithm is $\mathcal{H}_k = \ln k + \theta(1)$ (with $\mathcal{H}_k = \sum_{i=1}^{k} 1/i$ the k-th Harmonic number).

7.1 Semi-local Search for k-SET COVER

Halldórsson [52] proposed a polynomial time local search algorithm with a performance ratio of $\mathcal{H}_k - 11/42$ for k-SET COVER, with $k \geq 3$. Duh and Fürer [37] obtained a better performance ratio, namely $\mathcal{H}_k - 1/2$ for k-SET COVER, with $k \geq 3$. They have proposed a local search based algorithm, they call semi-local search, for this problem. We consider first the 3-SET COVER problem. A semi-local (s, t)-improvement step consists of the insertion of up to s 3-sets, and the deletion of up to t 3-sets from a current solution. Moreover an arbitrary number of 2-sets and 1-sets are optimally replaced. Indeed, once the 3-sets have been selected for a partial cover, the task of selecting the 2-sets and 1-sets can be done optimally in polynomial time by computing a maximum matching. The vertices are the elements of U still to cover and the edges are the 2-sets. The 2-sets corresponding to the maximum matching edges and the 1-sets corresponding to the vertices not covered by the maximum matching edges form a optimum covering.

This is an example of non-oblivious local search. The objective function used in the local search is not the number of sets in the cover but also takes into account the number of 1-sets. More precisely these two quantities (total number of sets in the cover and number of 1-sets) are ordered in a lexicographic way. We prefer smaller covers, and among covers of a given size we prefer those with fewer 1-sets.

Theorem 17 (Duh, Fürer, 1997[37]). *The semi-local $(2, 1)$-search algorithm for* 3-SET COVER *produces a solution with performance ratio 4/3, and this bound is tight.*

Proof. (sketch) Duh and Fürer introduce the following bipartite multi-graph. It has an A-vertex for every set in the local search solution A and it has a B-vertex for every set present in a fixed optimal (best) solution B. The element of the base set U are represented by the edges. If the set corresponding to a A-vertex intersects the set corresponding to a B-vertex in m elements, then there are m edges between the two vertices. Let A_j (resp. B_j) be the set of A-vertices (resp. B-vertices) of degree j. We note $|A_j| = a_j$, $|B_j| = b_j$ for $j = 1, 2, 3$, $|A| = a$, $|B| = b$ and $|U| = u$.

Since we have assumed that the collection of subsets was closed under subsets, we can assume without loss of generality that the sets of the cover in solution A are disjoint. The same for solution B. Then it is possible to prove that

$$a_1 + 2a_2 + 3a_3 = b_1 + 2b_2 + 3b_3 = |U|, \tag{8}$$

$$a_1 \leq b_1, \text{ and} \tag{9}$$

$$a_1 + a_2 \leq b_1 + b_2 + b_3. \tag{10}$$

Adding the equality 8 and inequalities 9,10 we get $3(a_1 + a_2 + a_3) \leq 3b_1 + 3b_2 + 4b_3$, hence

$$3a \leq 4b - b_1 - b_2 \leq 4b, \tag{11}$$

and $a \leq \frac{4}{3}b$.

The example in Figure 4 shows that this bound is tight. □

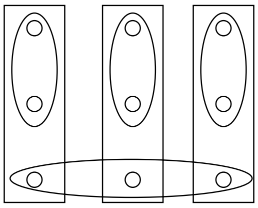

Fig. 4. Worst case for 3-SET COVER for semi-local $(2, 1)$-search algorithm

Notice that with the greedy algorithm we obtain $\mathcal{H}_3 = 1 + \frac{1}{2} + \frac{1}{3} \simeq 1.83 > 4/3$.

We consider now the k-SET COVER problem. A semi-local (s, t)-improvement step consists of the insertion of up to s 3-sets, 4-sets, ..., k-sets and the deletion of up to t such sets from a current solution. By using a similar proof as in Theorem 17 it is possible to prove that the semi-local $(2, 1)$-search algorithm for k-SET COVER problem achieves a performance ratio of $(k + 1)/3$, which is not competitive with the performance ratio obtained by the greedy algorithm when k gets larger. In order to improve this result Duh and Fürer propose an algorithm which combines a greedy phase followed by a semi-local search. The aim of the greedy phase is to select the l-sets with $l \geq 5$, then the local search is used to select the 4-sets and 3-sets to cover the remaining elements (recall that the 2-sets and 1-sets are optimally chosen by using matching techniques when all other subsets have been chosen).

Theorem 18 (Duh, Fürer, 1997[37]). *The semi-local* $(2,1)$*-search algorithm for* k-SET COVER *produces a solution with performance ratio* $\mathcal{H}_5 - 5/12$ *for* $k \geq 4$.

By using a more complex strategy it is possible to improve this result and obtain a performance ratio $\mathcal{H}_5 - 1/2$ for $k \geq 4$ [37].

7.2 Application for COLOR SAVING

In the color saving problem one seeks to color a graph with the minimum number of colors. The objective function to maximize is the total number of vertices n minus the number of colors used.

Theorem 19 (Duh, Fürer, 1997[37]). *The semi-local* $(2,1)$*-search algorithm has a performance ratio* $6/5$ *for* COLOR SAVING *on graphs with maximum independent sets of size 3.*

Proof. Under this assumption COLOR SAVING is a special case of 3-SET COVER. The base set U is the set of vertices of the given graph, and the collection C is all independent sets of size at most 3.

We have $u = b_1 + 2b_2 + 3b_3$ vertices and the number of colors used is $b = b_1 + b_2 + b_3$ for a best solution B and $a = a_1 + a_2 + a_3$ for a solution A returned by the local search algorithm. Therefore the value of the objective function is $u - a$ (resp. $u - b$) for solution A (resp. B).

By inequality 11, $3a \leq 4b - b_1 - b_2$, and so

$$3u - 4b + b_1 + b_2 \leq 3(u - a). \tag{12}$$

With $b = b_1 + b_2 + b_3$ and $u = b_1 + 2b_2 + 3b_3$ we have

$$3b - 2b_1 - b_2 - u = 0. \tag{13}$$

Multiplying inequality 12 by 2 and adding equality 13 we obtain $5u - 5b \leq 5u - 5b + b_2 \leq 6(u - a)$. Therefore $(u - b)/(u - a) \leq \frac{6}{5}$. □

In the general case, i.e. when larger independent sets exist, Duh and Fürer [37] obtain an algorithm with a performance ratio $360/289 \simeq 1.2456$.

Local search has also been applied to a general graph coloring problem but with less success. Given an edge-weighted graph and an integer k, the generalized graph coloring problem is the problem of partitioning the vertex set into k subsets so as to minimize the total weight of the edges that are included in a single subset. Vredeveld and Lenstra [101] show that the quality of local optima with respect to a large class of neighborhoods may be arbitrarily bad and that some local optima may be hard to find because of PLS-completeness.

8 The Maximum Cut Problem

Let $G = (V, E)$ be an undirected graph without loops and multiple edges. For any subsets $X, Y \subseteq V$ we denote by $[X, Y]$ the set of edges of G having one vertex in X and the other vertex in Y. The maximum cut problem, denoted by MAX CUT, is to find a bipartition (A, B) of the vertices of G, i.e. $A \cup B = V$ and $A \cap B = \emptyset$, such that $|[A, B]| = |\{x, y\} \in E : x \in A, y \in B\}|$ is maximized. In the directed case, the objective is to maximize the number of arcs whose heads are in B and whose tails are in A. This problem is denoted MAX DICUT.

In the sequel we consider the MAX CUT problem. This problem is NP-complete [68]. There exist very strong approximation results based on semi-definite programming [46,47] obtained by Goemans and Williamson. Feige, Karpinski and Langberg [42] show that a local search algorithm improves the performance guarantee of the Goemans and Williamson's algorithms [46,47].

Let $C_{opt}(G)$ be the optimal cost of MAX CUT on graph G. Edwards [38,39] has proved that $C_{opt}(G) \geq \frac{1}{2}|E| + \frac{1}{8}(\sqrt{8|E|+1} - 1)$, and that $C_{opt}(G) \geq \frac{1}{2}|E| + \frac{1}{4}(|V| - 1)$ for every connected graph. In the sequel we will see that local search is strong enough to attain the first bound, but not strong enough to obtain a bound as good as the latter inequality.

Neighborhoods for MAX CUT are defined by switching the location of the vertices of a subset $W \subset V$ in a given bipartition (A, B). It gives rise to a new bipartition $(A', B') = (A \Delta W, B \Delta W)$, where Δ denotes the symmetric difference of sets. Given a set of switchings S, we define the S-switch neighborhood of bipartition (A, B) as the set of bipartitions $(A', B') = (A \Delta W, B \Delta W)$ for $W \in S$. When $S = V$, i.e. we can switch a single vertex from one subset to the other, we have the flip neighborhood. It is easy to see that a local search algorithm with respect to the flip neighborhood achieves a performance ratio of 2 [100]. It has been proved that MAX CUT with the flip neighborhood is PLS-complete [94]. However this is no more true when one considers cubic graphs [90] (the problem remains NP-hard).

In the sequel we shall note $d(X, Y) = |[X, Y]|$, and we simply write $d(v, Y)$ instead of $d(\{v\}, Y)$ if $v \in V$. We have $d(v, w) = 1$ if $\{v, w\} \in E$ and $d(v, w) = 0$ otherwise. Bylka, Idzik and Tuza [25] define the following quantities: For every bipartition (A, B) of V, the *weight* of the vertex v with respect to the bipartition (A, B) is $\mu(v, A, B) = d(v, B) - d(v, A)$ if $v \in A$ and $\mu(v, A, B) = d(v, A) - d(v, B)$ if $v \in B$. The *measure of effectiveness* of the bipartition (A, B) is $\overline{\mu}(A, B) = \sum_{v \in V} \mu(v, A, B)$. It is easy to see that for every bipartition (A, B) of the graph $G = (V, E)$ one has

$$|[A, B]| = \frac{1}{2}|E| + \frac{1}{4}\overline{\mu}(A, B). \tag{14}$$

In the following we will write $A \cup w$ instead of $A \cup \{w\}$, and so on.

If we restrict ourself to set W of size at most 2, then it is easy to derive that $\overline{\mu}(A', B') > \overline{\mu}(A, B)$ if and only if, either

$$W = \{v\} \text{ and } \mu(v, A, B) < 0, \text{or} \tag{15}$$

$$W = \{v, w\} \text{ and } \mu(v, A, B) + \mu(w, A, B) < 2d(v, w). \tag{16}$$

For any bipartition (A, B), Bylka, Idzik and Tuza define the following sets of switchings:

$$\begin{aligned}
S_1(A,B) &= \{v \in V \mid \mu(v,A,B) < 0\}, \\
S_2(A,B) &= \{v \in V \mid \mu(v,A,B) < 0 \text{ and } |A - v| = |B - v|\} \cup \\
&\quad\ \{v, w \in V^2 \mid \mu(v,A,B) + \mu(w,A,B) < 2d(v,w)\}, \\
S_3(A,B) &= S_1(A,B) \cup \{v \in B \mid \mu(v,A,B) = 0\} \cup \\
&\quad\ \{\{v,w\} \in [A,B] \mid \mu(v,A,B) + \mu(w,A,B) < 2\}.
\end{aligned}$$

Because of conditions 15 and 16 these sets of switchings define improving moves. Each time a switch is performed the value $\overline{\mu}(A, B)$ increases by at least 4 and therefore the number of iterations performed by each local search is $\mathcal{O}(|E|)$. Moreover the size of the S_1,S_2 and S_3-switch neighborhoods are respectively $\mathcal{O}(|V|)$, $\mathcal{O}(|V|^2)$ and $\mathcal{O}(|V| + |E|)$, leading to polynomial time local search algorithms.

Theorem 20 (Bylka, Idzik, Tuza, 1999[25]). *For any graph $G = (V, E)$ and any (balanced for the S_2-switch neighborhood) partition (A, B) which is a local optimum for respectively the S_1 S_2 and S_3-switch neighborhood, one has for the size of the cut $[A, B]$ respectively*

$$|[A,B]| \geq \frac{1}{2}|E| + \frac{1}{4}s(G),$$

where $s(G)$ is the number of vertices having odd degree,

$$|[A,B]| \geq \frac{1}{2}|E|\frac{|V| + \varepsilon}{|V| - 1 + \varepsilon},$$

with $\varepsilon = 0$ if $|V|$ is even and $\varepsilon = 1$ if $|V|$ is odd,

$$|[A,B]| \geq \frac{1}{2}|E| + \frac{1}{8}(\sqrt{8|E| + 1} - 1).$$

Moreover these three bounds are sharp.

Proof. (sketch) Since (A, B) is a local optimum for the S_1-switch neighborhood one has $\forall v \in V$, $\mu(v, A, B) \geq 0$. If the degree of v is odd it implies that $\mu(v, A, B) \geq 1$, and by equality 14 one gets $|[A, B]| \geq \frac{1}{2}|E| + \frac{1}{4}s(G)$.

Since (A, B) is a local optimum for the S_2-switch neighborhood one has $\forall (v, w) \in A \times B$,

$$\mu(v,A,B) + \mu(w,A,B) \geq 2d(v,w) = \begin{cases} 2 \text{ if } \{v,w\} \in E \\ 0 \text{ if } \{v,w\} \notin E \end{cases}$$

Therefore summing this inequality over all $(v, w) \in A \times B$ we get

$$|B| \sum_{v \in A} \mu(v,A,B) + |A| \sum_{w \in B} \mu(w,A,B) \geq 2|[A,B]|. \tag{17}$$

Since the bipartition (A, B) is balanced we can assume that $|B| \leq |A| \leq |B|+1$, and $|A| = \frac{1}{2}|V| + \frac{1}{2}\varepsilon$ and $|B| = \frac{1}{2}|V| - \frac{1}{2}\varepsilon$. Therefore, since (A, B) is a local optimum for the S_2-switch neighborhood one has $\mu(v, A, B) \geq 0$ for $v \in A$. From the inequality 17 we get

$$
\begin{aligned}
2|[A,B]| &\leq \frac{1}{2}\,|V|\,\overline{\mu}(A,B) + \frac{1}{2}\,\varepsilon\,(\sum_{w\in B}\mu(w,A,B) - \sum_{v\in A}\mu(v,A,B)) \\
&\leq \frac{1}{2}\,|V|\,\overline{\mu}(A,B) + \frac{1}{2}\,\varepsilon\,\overline{\mu}(A,B) = \frac{1}{2}\,(|V|+\varepsilon)\,\overline{\mu}(A,B).
\end{aligned}
$$

By equality 14 we obtain $\frac{1}{2}\,(|V|+\varepsilon)\,\overline{\mu}(A,B) \geq |E| + \frac{1}{2}\,\overline{\mu}(A,B)$ and the result follows.

The third result is more difficult to get, and we refer the reader to the paper [25] for the details. □

The following theorem shows that (oblivious) local search cannot attain the second bound of Edwards.

Theorem 21 (Bylka, Idzik, Tuza, 1999[25]). *For every natural number k there exist a real ε_k, a constant c_k, and an infinite family of connected graphs $G_n = (V_n, E_n)$ with vertex partition (A_n, B_n) such that $|[A_n, B_n]| \leq \frac{1}{2}|E_n| + (\frac{1}{4} - \epsilon_k)\,(|V_n| - 1) + c_k$ holds, and no switching of at most k vertices can increase the number of edges in the bipartition.*

9 Facility Location Related Problems

In this class of problems we have a set of facilities F and a set of clients C. We are given a cost (or distance) c_{ij} which represents the cost of serving client $j \in C$ from facility $i \in F$. The objective is to determine a subset of facilities $S \subseteq F$ to open in order to serve all the clients at the minimum cost. In the metric version, we assume that the costs c_{ij} are symmetric ($F \cap C$ is not necessarily the empty set) and satisfy the triangle inequality. We will consider the following well known problems.

In the uncapacited k-median problem, denoted by UKM, we want to open at most k facilities, i.e. $|S| \leq k$. Let us assume that a client $j \in C$ is served by a facility $\sigma(j) \in F$. Then we want to minimize the total cost of serving all clients: $cost(S) = cost(S, \sigma) = \sum_{j\in C} c_{\sigma(j)j}$. Notice that when S is chosen, serving each client by its nearest facility in S, i.e. $\sigma(j) = \arg\min_{l\in S} c_{lj}$, minimize this cost. One sometimes assumes that each client $j \in C$ has a demand d_j. In that case the cost is $cost(S) = \sum_{j\in C} d_j c_{\sigma(j)j}$. In the capacited k-median problem there is an upper bound on the demand that can be satisfied by any given facility. There are two variants to consider: either the demand of a client must be met by a single facility (unsplittable demands), or the demand of a client may be divided across several facilities (splittable demands). The capacited k-median problem with unsplittable demands is denoted by CKMU, whereas the capacited k-median problem with splittable demands is denoted by CKMS. For these problems, an

(a, b)-approximation algorithm is a polynomial time algorithm that computes a solution using at most bk facilities and with a cost at most a times the cost of an optimal solution using at most k facilities.

In the k-center problem we want to open k facilities, such that the maximum serving cost is minimized, i.e. we want to minimize $\max_{j \in C} c_{\sigma(j)j}$. Hassin, Levin and Morad [56] show that a lexicographic variant of local search achieves some performance ratio for this problem but on very specific cases ($k = 2$ or $k = 3$).

In the uncapacited facility location problem, denoted by UFL, we are given costs $f_i \geq 0$ for opening the facility $i \in F$. The problem is to determine a subset of open facilities $S \in F$ in order to minimize the cost of opening facilities plus the cost of serving all clients from these opened facilities: $cost(S) = \sum_{i \in S} f_i + \sum_{j \in C} c_{\sigma(j)j}$. As in the UKM problem, when the subset S is chosen, serving each client by its nearest facility in S minimizes the serving cost. As before capacited versions can be considered. We denote by CFLU the capacited facility location problem with unsplittable demands, and by CFLS the capacited facility location problem with splittable demands. In these problems at most one facility can be opened at a given location.

Guha, Meyerson and Munagala [50] consider a generalization of the classical facility location problem, where solutions are required to be fault-tolerant. Every client point j must be served by r_j facilities instead of just one. The cost of assignment for client j is the weighted linear combination of the assignment costs to its r_j closest open facilities. The objective is to minimize the sum of the cost of opening the facilities and the assignment cost of each client j. Guha, Meyerson and Munagala obtain a factor 4 approximation to this problem through the application of various rounding techniques to the linear relaxation of an integer program formulation. They further improve the approximation ratio to 3.16 using randomization and to 2.41 using greedy local-search type techniques.

For a capacited problem with splittable demands, when a subset S is chosen an optimal assignment of clients to facilities can be computed in polynomial time by solving a transportation problem. However, for a capacited problem with unsplittable demands, when a subset S is chosen finding an optimal assignment of clients to facilities is NP-hard. The results presented in [74] use a reduction from the unsplittable case to the splittable case.

In another variant of the capacited facility location problem, denoted by CFL, we associate a capacity u_i with each facility $i \in F$, which measures the maximum number of clients that the facility can serve. We can open multiple copies of a facility i. Each copy incurs a cost f_i and is able to serve at most u_i clients. The objective function is the same than before, $cost(S) = \sum_{i \in S} f_i + \sum_{j \in C} c_{\sigma(j)j}$. When the subset S is chosen, minimizing the serving cost $\sum_{j \in C} c_{\sigma(j)j}$ can be done efficiently by a mincost flow problem. In the k-CFL we furthermore bound the number of facilities that than be opened at a certain location, namely we can open at most k facilities at any location.

For UKM, Arya et al. [10] show that a local search algorithm with the swap neighborhood has a performance ratio of 5 and this bound is tight. When p facilities are allowed to be swapped simultaneously, that is we can drop at most

p facilities and open the same number of new facilities, the performance ratio becomes $3 + 2/p$, and it is tight. Korupolu, Plaxton and Rajaraman [74] give a local search procedure in which we can add, delete and swap facilities. The algorithms returns a solution with $k(1+\epsilon)$ opened facilities having a service-cost at most $3 + 5/\epsilon$ times the optimum k-median solution.

For UFL, Arya et al. [10] show that a local search in which we can add, drop and swap a facility, has a performance ratio of 3 and this bound is tight. Using scaling techniques [30] the algorithm can be improved to obtain a ratio of $1+\sqrt{2} \simeq 2.414$. The best approximation ratio is achieved by the 1.728-approximation algorithm due to Charikar and Guha [30] that uses a hybrid approach by combining an LP-based approximation algorithm with a local improvement algorithm in which multiple facilities may by dropped in a single local search step.

For ∞-CFL, Arya et al. [10] show that a local search in which we can add multiple copies of a facility and drop zero or more facilities has a performance ratio of 4. Jain and Vazirani [64] obtained the same ratio using a primal-dual approach. Using scaling techniques [30] the algorithm of Arya et al. can be improved to obtain a ratio of $2 + \sqrt{3} \simeq 3.732$.

Kanungo et al. [67] consider the k-means clustering problem. We are given a set of n data points in d-dimensional space $\mathbb{R}$ and an integer k, and the problem is to determine a set of k points in $\mathbb{R}^n$, called centers, to minimize the mean squared distance from each data point to its nearest center. Although asymptotically efficient approximation algorithms exist [82], these algorithms are not practical due to the extremely high constant factors involved. Kanungo et al. present a local improvement heuristic based on swapping centers in and out. They prove that this yields a $(9 + \epsilon)$-approximation algorithm, and that the approximation factor is almost tight by giving an example for which the algorithm achieves an approximation factor of $(9 - \epsilon)$.

In the following we will present only a sketch of proof of a result obtained by Arya et al.:

Theorem 22 (Arya, Meyerson, Pandit, Garg, Munagala, Khandekar, 2001[10]). *For the uncapacited k-median problem* UKM*, a local search with the swap neighborhood achieves a performance ratio of 5.*

Proof. (sketch) In order to obtain a polynomial time local search algorithm, one moves from a solution S to a neighboring solution S' only if the decrease in cost is large enough, i.e. $cost(S') \leq (1 - \frac{\epsilon}{p(|F|,|C|)})cost(S)$, with $\epsilon > 0$ and $p(|F|, |C|)$ is some polynomial in the number of facilities and clients. It can be shown that by doing this, one gets a $5(1 + \epsilon)$ approximation ratio instead of 5. To simplify the exposition, we shall assume that the algorithm ends at a solution S such that $cost(S') \geq cost(S)$ for any neighboring solution S'.

Let S denote the output of the local search algorithm, and S^* denote an optimum solution. Since S is a local optimum with respect to the swap neighborhood, one has $cost(S - s + o) \geq cost(S)$, $\forall s \in S, o \in S^*$. By combining some of these inequalities it is possible to get that $cost(S) \leq 5\, cost(S^*)$. □

10 Scheduling Problems

We consider some classical and extensively studied scheduling problems. We have a set of n jobs, $J_1, \ldots, J_n$, which has to be processed without preemption on machines $M_1, \ldots, M_m$. A machine can process at most one job at a time. The processing time of job J_j if it is scheduled on machine M_i is denoted by p_{ij}. One has to determine for each job the machine on which it is scheduled. The load of a machine is the sum of the processing times of jobs scheduled on it. The objective is to find a schedule of minimum makespan, i.e. a schedule which minimizes the maximum load of machines.

In the identical parallel machines environment, denoted by P, any job J_j has the same processing time p_j on all machines, i.e. $p_{ij} = p_j \; \forall i \in \{1, \ldots, m\}$. In the uniform parallel machines environment, denoted by Q, each machine M_i has a speed s_i, each job J_j has a processing requirement p_j, and the processing time of job J_j if it is scheduled on machine M_i is given by $p_{ij} = p_j / s_i$. In the unrelated parallel machines environment, denoted by R, p_{ij} depends both on job J_j and machine M_i. In the classical notation of Graham et al. [48], when the number of machines is part of the input, these problems are denoted by respectively $P||C_{max}$, $Q||C_{max}$ and $R||C_{max}$. When the number of machines is a constant m, these problems are denoted by respectively $Pm||C_{max}$, $Qm||C_{max}$ and $Rm||C_{max}$. All these problems are NP-hard, since even the simplest case, $P2||C_{max}$ is NP-hard [45].

For $P||C_{max}$ and $Q||C_{max}$ problems, Hochbaum and Shmoys [60,61] proposed a polynomial-time approximation scheme, that is, for each $\epsilon > 0$, there exists a polynomial time $(1+\epsilon)$-approximation algorithm. For $R||C_{max}$, Lenstra, Shmoys and Tardos [78] proposed a polynomial time 2-approximation algorithm. They also proved that there does not exist a polynomial time $(3/2 - \epsilon)$-approximation algorithm for $\epsilon > 0$, unless P=NP.

Schuurman and Vredeveld [95] analyze the performance of local search algorithms for theses problem. Despite their results is not competitive with the above mentioned ones, they are nevertheless interesting since they constitute one of the very few results concerning the worst case performance analysis of local search algorithms for scheduling problems. They consider three neighborhoods, based on jump, swap and push moves. In the *jump* (also called *move*) neighborhood, we can move a job to a machine on which it is not scheduled. Let us define a critical machine as a machine with maximum load. Then, a local optimum solution is a solution such that no jump decreases the makespan or the number of critical machines without increasing the makespan. In the *swap* neighborhood, we can interchange the machine allocations of any two jobs which are not scheduled on the same machine. If all jobs are scheduled on the same machine, then this neighborhood becomes empty. Therefore, the swap neighborhood consists of all possible jumps and all possible swaps.

The push neighborhood is the more complex one. It is based on the same idea than the variable-depth search introduced by Kernighan and Lin for the graph partitioning problem [69], and the traveling salesman problem [80]. A push is a sequence of jumps. It is initiated by selecting a job J_k on a critical machine, i.e.

with load C_{max}, and a machine M_i to move it. If after this move the load of machine M_i is at least as large as the original makespan C_{max} then the smallest job (among those which have a processing times smaller than the processing time of J_k) from M_i are iteratively removed from it, until the load of machine M_i becomes smaller than C_{max}. If it is possible the job J_k is said to fit on machine M_i. The removed jobs are gathered in a queue. Then the largest job of the queue are removed, and pushed in the same way than the job J_k, and so on. If at any time a job does not fit on any machine, then the push is unsuccessful. When pushing all jobs on the critical machine is unsuccessful, we have a local optimum solution with respect to the push neighborhood. It can be shown that a push move can be done, using appropriate data structures and selecting, in $\mathcal{O}(n \log n)$ time. Notice that in the case of unrelated parallel machines a push is not defined, since the largest job of the queue is not defined any more since its processing time depends on the machine it is assigned.

Table 1 resumes the results obtained by Schuurman and Vredeveld [95]. The performance guarantees for the jump neighborhood can be derived from the proof of Cho and Sahni [31] for list schedules on uniform parallel machines.

Table 1. Performance guarantees ρ of local search (from [95])

	jump	swap	push
$P2\|\|C_{max}$	$4/3$	$4/3$	$8/7$
$P\|\|C_{max}$	$2-\frac{2}{m+1}$	$2-\frac{2}{m+1}$	$\frac{4m}{3m+1} \le \rho \le 2-\frac{2}{m+1}$
$Q2\|\|C_{max}$	$\frac{1+\sqrt{5}}{2}$	$\frac{1+\sqrt{5}}{2}$	$\frac{1+\sqrt{17}}{4}$
$Q\|\|C_{max}$	$\frac{1+\sqrt{4m-3}}{2}$	$\frac{1+\sqrt{4m-3}}{2}$	$\frac{3}{2}-\epsilon \le \rho \le 2-\frac{2}{m+1}$
$R2\|\|C_{max}$	$\frac{p_{max}}{C_{opt}} \le \rho$	$n-1 \le \rho$	undefined
$R\|\|C_{max}$	$\frac{p_{max}}{C_{opt}} \le \rho$	$\frac{p_{max}-1}{C_{opt}} \le \rho$	undefined

Theorem 23 (Schuurman and Vredeveld, 2001[95]). *A push optimal solution for $P_2||C_{max}$ has value at most 8/7 times the optimal solution value.*

Proof. (sketch) Let L_i $(i = 1, 2)$ the load of machine M_i, and assume w.l.o.g. that $L_1 \geq L_2$. Then it can be shown that $L_1 - L_2 > L_1/4$. Let J_1 be the smallest job on M_1. By push optimality $p_1 \leq L_1 - L_2 > L_1/4$. Hence there are at most three jobs on M_1. By reasoning on the largest job that has to be removed when pushing J_1 onto M_2, and by careful enumerations of different cases according the number of jobs that are processed on machine M_2, Schuurman and Vredeveld derive the theorem. □

Brucker, Hurink and Werner [21,22] show that for $P2||C_{max}$ and $P||C_{max}$ a local optimum with respect to the jump neighborhood can be found after $\mathcal{O}(n^2)$ iterations. This result can be improved for $P2||C_{max}$. If deepest descent local search is used, then a local optimum solution can be found after $\mathcal{O}(n)$ iterations. Schuurman and Vredeveld obtain for $Q2||C_{max}$ the same result. For $Q||C_{max}$ a local optimum solution can be found after $\mathcal{O}(n^2 m)$ iterations.

However it is an open problem to determine the number of iterations needed to find a local optimum solution with respect to the swap or push neighborhood. Schuurman and Vredeveld [95] conjecture that a push local optimum solution cannot be found in polynomial time via an iterative improvement procedure.

11 Minimum Label Spanning Tree Problem

In the minimum label spanning tree problem, denoted by MIN LST, we have an undirected, connected graph G. The edges of G are colored (or labeled) with the colors $c_1, c_2, \dots, c_q$. The goal of the MIN LST problem is to find a spanning tree of G that uses the minimum number of colors, or equivalently to find a smallest cardinality subset $C \subseteq \{c_1, c_2, \dots, c_q\}$ of the colors, such that the subgraph induced by the edges with colors belonging to C is connected and touches all vertices of G.

This problem, introduced by Chang and Leu [29] in the context of communication network design, is NP-hard. Krumke and Wirth [76] proved that it is not approximable better than a logarithmic factor, unless P=NP, and provided a greedy algorithm with an approximation ratio of at most $2 \ln n + 1$. Wan, Chen and Xu [102] obtained an approximation ratio of at most $\ln(n-1)+1$. Brüggemann, Monnot and Woeginger [23] have studied a special case, denoted by MIN LST$_r$, where every color can occur at most r times on the edges of graph G. They show that for $r = 2$ this problem can be solved in polynomial time, since it is equivalent to the graphic matroid parity problem. For any fixed $r \geq 3$ they show that this problem is APX-complete, which implies that it does not have a polynomial time approximation scheme unless P=NP. For $r \geq 3$ they introduce a local search algorithm.

Let $C \subseteq \{c_1, c_2, \dots, c_q\}$ be a color set. The color set C is said to be feasible if the set of edges of G with colors belonging in C is connected and touches all vertices of G. Let C_1 and C_2 be two color sets. The color set C_2 is in the k-switch neighborhood of the color set C_1 if and only if we can get C_2 by first removing up to k colors from C_1, and then adding up to k other colors to it. The local search algorithm looks, at each iteration, if there exists a feasible color set C' in the k-swith neighborhood of the current solution C such that $|C'| < |C|$, i.e. C' uses strictly less colors than C. A spanning tree is said to be a local optimum for the k-switch neighborhood if its associated color set (the set of colors of its edges) is a local optimum for the k-switch neighborhood.

Theorem 24 (Brüggemann, Monnot, Woeginger, 2003[23]). *For any integer $r \geq 2$ and for any instance of* MIN LST$_r$*, the objective value of any local optimum with respect to the 2-switch neighborhood is at most a factor $(r+1)/2$ above the optimal objective value, and this bound is tight.*

Proof. (sketch) The proof is by contradiction. Let $G = (V, E)$ be a counterexample with the smallest number of edges. Let $T^* = (V, E^*)$ be an optimal spanning tree for G, with color set C^*, and let $T^+ = (V, E^+)$ be a locally optimal tree with respect to the 2-switch neighborhood, with color set C^+. We have therefore $|C^+| > \frac{r+1}{2}|C^*|$.

A color is said to be a singleton if it appears on exactly one edge of G. Let l denote the number of singleton colors in C^+, and let $e_1, e_2, \ldots, e_l$ be the corresponding edges in T^+. A central step in the proof is to show that $|C^*| \geq l$. The main idea is to consider the $l+1$ subtrees $T_1^+, \ldots, T_{l+1}^+$ that result from removing the l edges $e_1, e_2, \ldots, e_l$ from T^+. There does not exist some color c_i, such that the edges with color c_i connect more than two of these subtrees to each other, otherwise C^+ would not be a local optimum with respect to the 2-switch neighborhood. Therefore every color connects at most two of these $l+1$ trees to each other. It implies that the global optimum must use at least l colors for connecting the corresponding $l+1$ vertex set $T_1^+, \ldots, T_{l+1}^+$ to each other, and therefore $|C^*| \geq l$. It is easy to show that the inequality $|C^*| \geq (n-1)/r$ also holds, and these two inequalities, together with some others arguments, are used to show that $|C^+| \leq \frac{r+1}{2}|C^*|$, contradicting the initial assumption. One can construct instances which show that this bound is tight (see [23]). □

From Theorem 24 and using observations from the paper [23] we get the following corollary.

Corollary 3. *The local search algorithm with 2-switch neighborhood is a polynomial time* $(r+1)/2$*-approximation algorithm for* MIN LST$_r$.

Proof. The size of the k-switch neighborhood of a color set C is at most $\mathcal{O}(|C|^k q^k)$, which is at most $\mathcal{O}(n^{3k})$ since $|C| \leq n-1$ and the number of colors q is at most the number of edges of graph G, i.e. $q \leq n^2$. Moreover it takes $\mathcal{O}(n^2)$ time to determine if a color set in the neighborhood of C is feasible and determine its objective value. Therefore, the exploration of the neighborhood takes $\mathcal{O}(n^{3k+2})$ time. Moreover the number of steps to reach a local optimum solution is at most $n-2$, since the objective value is an integer between 1 and $n-1$. □

Brüggemann, Monnot and Woeginger [23] show that there is almost no profit (from the worst case point of view) in moving from the small 2-switch neighborhood to a larger one such as k-switch with $k \geq 3$. Indeed, for every $k \geq 3$, there exist instances for which some local optimum, with respect to the k-switch neighborhood, is a factor of roughly $r/2$ away from the global optimum.

12 Replica Placement in a Distributed File Systems

In [35] Douceur and Wattenhofer consider a distributed file systems in which each file is stored multiple times on different machines to maximize its availability. A file is not available if all machines that store the replicas of the file are down in the same time. The availability is expressed as the negative logarithm of fractional downtime, and therefore the availability of a file is equal to the sum of the availabilities of the machines that store the file's replicas. The aim is to maximize the minimum availability of a file. Douceur and Wattenhofer propose simple online algorithms based on local search. Their objective is to have a simple and efficient distributed algorithm, independently of the NP-hardness or polynomial solvability of the underlying problem.

The problem can be defined as follows. Let us consider N files, each of which must be stored on R (a constant) different machines. Each machine has the capacity to store only a single file. We have a set $\mathcal{P}$ of $M = NR$ machines availabilities. Let P_f denote the multiset of availabilities of the R machines which store the R replicas of file f. Then the availability of a file f is $\sum_{a \in \mathcal{P}_f} a$. The sets $\mathcal{P}_1, \ldots \mathcal{P}_N$ constitute a partition of $\mathcal{P}$. The problem if to find a partition in order to maximize the minimum availability of a file $\min_{f=1,\ldots N} \sum_{a \in \mathcal{P}_f} a$.

The local search algorithm start from an arbitrary partition and make improvements until reaching a local optimal solution. The neighborhood consist in swapping two replicas files, i.e. the machines in which two replicas (of different files) are stored are exchanged.

Three variants are considered. In the *MinMax* neighborhood the only allowed replica-location swaps are between the file with the minimum availability and the file with the maximum availability. In the *MinRand* neighborhood the only allowed swaps are between the file with the minimum availability and any other file. In the *RandRand* neighborhood swaps are allowed between any pair of files. Notice that the size of these neighborhoods grows exponentially with R, but in practice only small values are used (typically $R = 2$ or 3).

The results obtained by Douceur and Wattenhofer are summarized in Table 2. This table shows the worst performance ratio of local search, over all possible availabilities of machines, when $N \to \infty$.

Table 2. The approximation ratio ρ obtained by local search

	MinMax	MinRand	RandRand
$R = 3$	$\rho = +\infty$	$\rho = 3/2$	$\rho = 3/2$
$R = 2$	$\rho = +\infty$	$\rho = 1$	$\rho = 1$
any R	$\rho = +\infty$	$\rho < 2$	$\rho \leq 17/10$

Notice that the MinMax neighborhood is included in the MinRand neighborhood, which is in turn included in the RandRand neighborhood. We have therefore $\rho_{MinMax} \geq \rho_{MinRand} \geq \rho_{RandRand}$.

To see that $\rho_{MinMax} = +\infty$ consider the following partition $\mathcal{P}_1 = \{0, 0, 0 \ldots, 0\}$, $\mathcal{P}_2 = \{3, 0, 0 \ldots, 0\}$ and $\mathcal{P}_i = \{1, 1, 0 \ldots, 0\}$ for $i = 3, \ldots N$. The availability of file 1 is $0 + \ldots + 0 = 0$, the availability of file 2 is 3, and other files have availabilities 2. We have therefore $m = \min\{0, 3, 2\} = 0$. This partition is a local optimum with the MinMax neighborhood, since the only allowed swaps are between files 1 and 2. However by performing a swap between machines 1 and 3 we can get the optimum partition $\mathcal{P}_1^* = \{1, 0, 0 \ldots, 0\}$, $\mathcal{P}_2^* = \{3, 0, 0 \ldots, 0\}$, $\mathcal{P}_3^* = \{1, 0, 0 \ldots, 0\}$ and $\mathcal{P}_i = \{1, 1, 0 \ldots, 0\}$ for $i = 4, \ldots N$. We have $m^* = 1$ and the result follows.

13 Summary and Conclusion

Table 3 shows the main results obtained by local search. One can see the various approximation ratios obtained by local search versus the best approximation ratio currently known for each problem. For MAX CUT and QAP problems the quantity given for local search denotes not an approximation ratio, but an absolute performance guarantee. One can see that for several problems the best ratio is obtained by using local search algorithms. Despite this table resumes a a vast amount of research, it is by no means exhaustive (see for example [84] for recent results concerning partial subgraphs problems which are not mentioned here, and [72] for results concerning the problem of finding a cotree (complement of a tree) incident upon the minimum number of vertices in a graph).

One fruitful area of research concerns the design of approximation algorithms for multicriteria optimization problems [40]. Such results have been obtained for

Table 3. Summary of main approximation results obtained by local search

problem	local search	ref	best known	ref
MAX k-SAT	$2^k/(2^k-1)$	[70, 71]	same as local search	[65]
MAX k-CCSP	2^k-1	[4]	2^{k-1}	[99]
MAX k-CUT	$\frac{k}{k-1}\frac{2\Delta+k-1}{2\Delta+k}$	[53]	1.21 for $k=3$ 1.18 for $k=4$ 1.15 for $k=5$	[43]
MAX 2-CCSP	2.5	[5]	1.165	[46, 41]
MAX DICUT	2.5	[5]	1.165	[46, 41]
MAX CUT	$C(s_{loc}) \geq \frac{1}{2}\|E\| + \frac{1}{8}(\sqrt{8\|E\|+1}-1)$	[25]	1.138	[47]
MAX CSP(k,r)	k/r	[53]	local search	[53]
MIN LST$_r$	$(r+1)/2$	[23]	local search	[23]
METRIC TSP	$4\sqrt{n}$	[28]	3/2	[32]
TSP(1,2)	3/2	[70, 71]	7/6	[88]
QAP	$C(s_{loc}) \leq \frac{\langle F,D\rangle_+}{s(F)s(D)}\, n\, C_{\text{AV}}$	[7]	not approximable	
k-SET PACKING	$k/2+\epsilon$	[63, 52]	local search	[63, 52]
W-k-SET PACKING	$(k+1)/2$	[17]	local search	[17]
k-SET COVER	$\mathcal{H}_k - 1/2$	[37]	local search	[37]
$P\|\|C_{max}$	$2-2/(m+1)$	[95]	$1+\epsilon$	[60, 61]
$Q\|\|C_{max}$	$2-2/(m+1)$	[95]	$1+\epsilon$	[60, 61]
$R\|\|C_{max}$	$\emptyset$	$\emptyset$	2	[78]
UKM	$(1+\epsilon, 3+5/\epsilon)$ $(1+5/\epsilon, 3+\epsilon)$ $3+2/p$	[74] [74] [10]	$(2(1+\epsilon), 1+1/\epsilon)$ $(2(1+\epsilon), 1+1/\epsilon)$ local search	[79] [79] [10]
UFL	2.414	[10]	1.728	[30]
CKMS	$(1+\epsilon, 5+5/\epsilon)$ $(1+5/\epsilon, 5+\epsilon)$	[74] [74]	local search local search	[74] [74]
CKMU with cap. blowup 2	$(1+\epsilon, 5+5/\epsilon)$ $(1+5/\epsilon, 5+\epsilon)$	[74] [74]	local search local search	[74] [74]
∞-CFL	3.732	[10]	local search	[10]
CFLS	$3+2\sqrt{2}+\epsilon$	[30]	local search	[30]
CFLU	$16+\epsilon$ (cap. blowup 2)	[74]	9 (cap. blowup 4)	[97]
k-MEANS CLUSTERING	$9+\epsilon$	[67]	$1+\epsilon$ (not practical)	[82]

facility location problems where (a, b)-approximation algorithms were defined. Other definitions of approximation are possible. Recently Angel, Bampis and Gourvès [6] have obtained approximation results for a bicriteria adapted local search procedure applied on a restricted case of the bicriteria traveling salesman problem. They show that the Pareto curve can be approximated within a ratio 3/2.

Often the neighborhoods are natural/simple but the analysis is highly specific to the problem. An attempt for unifying proofs would be very valuable. For example packing problems and set cover problems, which are in some sense "dual problems" have from the point of view of local search quite different proof techniques.

Non-oblivious local search should also deserves more consideration. In particular when combined with "non-oblivious neighborhoods", as the extended flip neighborhood for MAX 2-CCSP (see Theorem 4).

Finally, there are some gaps in the results. For example approximation results are known for W-k-SET PACKING, but to the knowledge of the author there are no such results for W-k-SET COVER. Another example is MAX DICUT for which no result is known for the neighborhood in which up to 2 vertices can be swapped, whereas there are results for the non directed case MAX CUT.

References

1. E. Aarts and J.K. Lenstra. *Local search in combinatorial optimization.* John Wiley and Sons, 1997.
2. R.K. Ahuja, Ö. Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.
3. P. Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. In *2nd Italian conference on algorithms and complexity*, LNCS 788, pages 40–53. Springer, 1994.
4. P. Alimonti. Non-oblivious local search for graph and hypergraph coloring problems. In *Graph-Theoretic Concepts in Computer Science, 21st International Workshop, WG '95, Aachen, Germany*, LNCS 1017, pages 167–180. Springer, 1995.
5. P. Alimonti. Non-oblivious local search for MAX 2-CCSP with application to MAX DICUT. In *Graph-Theoretic Concepts in Computer Science, 23rd International Workshop, WG '97, Berlin, Germany*, LNCS 1335, pages 2–14. Springer, 1997.
6. E. Angel, E. Bampis, and L. Gourvès. Approximating the pareto curve with local search for the bicriteria TSP(1,2) problem. *Theoretical Computer Science*, 310(1–3):135–146, 2004.
7. E. Angel and V. Zissimopoulos. On the quality of local search for the quadratic assignment problem. *Discrete Applied Mathematics*, 82:15–25, 1998.
8. E.M. Arkin and R. Hassin. On local search for weighted k-set packing. In *5th Annual European Symposium on Algorithms, ESA '97, Graz, Austria*, LNCS 1284, pages 13–22. Springer, 1997.
9. E.M. Arkin and R. Hassin. Approximating weighted set packing by local search. *Mathematics of Operations Research*, 23:640–648, 1998.

10. V. Arya, A. Meyerson, V. Pandit, N. Garg, K. Munagala, and R. Khandekar. Local search heuristics for k-median and facility location problems. In *ACM Symposium on Theory of Computing*, pages 21–29, 2001.
11. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.
12. G. Ausiello and M. Protasi. Local search, reducibility and approximability of NP optimization problems. *Information Processing Letters*, 54:73–79, 1995.
13. V. Bafna, B. Narayanan, and R. Ravi. Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles). *Discrete Applied Mathematics*, 71:41–53, 1996.
14. R. Battiti and M. Protasi. Solving MAX-SAT with non-oblivious functions and history-based heuristics. In *Satisfiability problems: Theory and applications*, DIMACS: Series in discrete mathematics and theoretical computer science, no. 35. AMS and ACM Press, 1997.
15. R. Battiti and M. Protasi. Approximate algorithms and heuristics for MAX-SAT. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of combinatorial optimization (vol. 1)*, pages 77–148. Kluwer, 1998.
16. C. Berge. *Graphs and Hypergraphs*. North Holland, Amsterdam, 1973.
17. P. Berman. A $d/2$ approximation for maximum weight independent set in d-claw free graphs. In *SWAT*, LNCS 1851, pages 214–219. Springer, 2000.
18. P. Berman and M. Furer. Approximating maximum independent set in bounded degree graphs. In *Proceedings of the fifth annual ACM-SIAM Symposium on discrete algorithms*, pages 365–371, 1993.
19. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
20. Y. Boykov, O. Veksler, and R. Zabih. A new algorithm for energy minimization with discontinuities. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, LNCS 1654, pages 205–220. Springer, 1999.
21. P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems I. *Discrete Applied Mathematics*, 65:97–122, 1996.
22. P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems II. *Discrete Applied Mathematics*, 72:47–69, 1997.
23. T. Brüggemann, J. Monnot, and G.J. Woeginger. Local search for the minimum label spanning tree problem with bounded color classes. *Operations Research Letters*, 31:195–201, 2003.
24. R.E. Burkard, E. Çela, S.E. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. http://www.seas.upenn.edu/qaplib/.
25. S. Bylka, A. Idzik, and Z. Tuza. Maximum cuts: Improvements and local algorithmic analogues of the Edwards-Erdös inequality. *Discrete Mathematics*, 194:39–58, 1999.
26. B. Chandra and M.M. Halldórsson. Greedy local improvement and weighted set packing approximation. In *SODA*, pages 169–176, 1999.
27. B. Chandra and M.M. Halldórsson. Greedy local improvement and weighted packing approximation. *Journal of Algorithms*, 39(2):223–240, 2001.
28. B. Chandra, H. Karloff, and C. Tovey. New results on the old k-opt algorithm for the TSP. *SIAM Journal on Computing*, 28(6):1998–2029, 1999.
29. R.-S. Chang and S.-J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63:277–282, 1997.

30. M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 378–388, 1999.
31. Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9:511–522, 1980.
32. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1976.
33. B. Codenotti and L. Margara. Local properties of some NP-complete problems. Technical Report TR-92-021, International computer science institute, Berkeley, April 1992.
34. E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.
35. J.R. Douceur and R.P. Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC), Lisbon, Portugal*, LNCS 2180, pages 48–62. Springer, 2001.
36. D.E. Drake and S. Hougardy. Linear time local improvements for weighted matchings in graphs. In K. Jansen, M. Margraf, M. Mastrolli, and J.D.P. Rolim, editors, *WEA 2003*, LNCS 2647, pages 107–119. Springer, 2003.
37. R. Duh and M. Fürer. Approximation of k-set cover by semi-local optimization. In *ACM Symposium on Theory of Computing*, pages 256–262, 1997.
38. C.S. Edwards. Some extremal properties of bipartite graphs. *Canadian Journal of Mathematics*, 25:475–485, 1973.
39. C.S. Edwards. An improved lower bound for the number of edges in a largest bipartite subgraph. In *Recent Advances in Graph Theory, Proc. Prague Symp., 1974, Academia, Praha*, pages 167–181, 1975.
40. M. Ehrgott. *Multicriteria Optimization.* Lecture Notes in Economics and Mathematical Systems, vol. 491. Springer, 2000.
41. U. Feige and M. Goemans. Approximating the value of the two prover proof system with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the 3rd Israeli Symposium on Theory of Computing and Systems*, pages 182–189, 1995.
42. U. Feige, M. Karpinski, and M. Langberg. Improved approximation of MAX-CUT on graphs of bounded degree. Technical Report 85215 CS, Institut für Informatik, Universität Bonn, 2000.
43. A. Frieze and M. Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, 18:67–81, 1997.
44. H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *SODA*, pages 434–443, 1990.
45. M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, New York, 1979.
46. M.X. Goemans and D.P. Williamson. .878-approximation algorithms for MAX-CUT and MAX 2SAT. In *Proceedings of the Twenty-Sixth ACM Symposium on the theory of computing*, pages 422–431. ACM, 1994.
47. M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

48. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
49. Lov K. Grover. Local search and the local structure of NP-complete problems. *Operations Research Letters*, 12:235–243, 1992.
50. S. Guha, A. Meyerson, and K. Munagala. A constant factor approximation algorithm for the fault-tolerant facility location problem. *Journal of Algorithms*, 48(2):429–440, 2003.
51. G. Gutin, A. Yeo, and A. Zverovitch. Exponential neighborhoods and domination analysis for the TSP. In G. Gutin and A.P. Punnen, editors, *Traveling salesman problem and its variations*, pages 223–256. Kluwer, 2002.
52. M.M. Halldórsson. Approximating discrete collections via local improvements. In *Proceedings of the sixth ACM-SIAM Symposium on Discrete Algorithms*, pages 160–169, 1995.
53. M.M. Halldórsson and H.C. Lau. Low-degree graph partitioning via local search with applications to constraint satisfaction, max cut, and 3-coloring. *Journal of Graph Algorithms and Applications*, 1(3):1–13, 1997.
54. M.M. Halldórsson and J. Radhakrishnan. Improved approximations of independent sets in bounded-degree graphs. In *Proceedings of the 4th Scandanavian Workshop on Algorithm Theory*, pages 194–206, 1994.
55. P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.
56. R. Hassin, A. Levin, and D. Morad. Lexicographic local search and the p-center problem. *European Journal of Operational Research*, 151:265–279, 2003.
57. J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.
58. E.A. Hirsch. Worst-case study of local search for MAX-k-SAT. *Discrete Applied Mathematics*, 130(2):173–184, 2003.
59. D.S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1982.
60. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.
61. D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. *SIAM journal on computing*, 17:539–551, 1988.
62. J. Hromkovic. *Algorithmics for hard problems.* Springer, 2002.
63. C.A. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.
64. K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 2–13, 1999.
65. D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.
66. D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37:79–100, 1988.
67. T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18, 2002.

68. R.M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–104. Plenum, New York, 1972.
69. B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291–307, 1970.
70. S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proceedings 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 819–836, 1994.
71. S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. Technical Report TR95-023, Electronic colloquium on computational complexity, 1995. http://www.eccc.uni-trier.de/eccc/.
72. S. Khuller, R. Bhatia, and R. Pless. On local search and placement of meters in networks. *SIAM journal on computing*, 32(2):470–487, 2003.
73. T.C. Koopmans and M. Beckmann. Assigment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.
74. M. Korupolu, C.G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of algorithms*, 37:146–188, 2000.
75. M.W. Krentel. Structure in locally optimal solutions. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 216–221, 1989.
76. S.O. Krumke and H.-C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66:81–85, 1998.
77. J. Lee. *A First Course in Combinatorial Optimization.* Cambridge University Press, 2004.
78. J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.
79. J.-H. Lin and J.S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44:245–249, 1992.
80. S. Lin and B.W. Kernighan. An effective heuristic for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
81. G. Lueker. manuscript, Princeton University, Princeton, NJ, 1975.
82. J. Matousek. On approximate geometric k-clustering. *Discrete and Computational Geometry*, 24:61–84, 2000.
83. J. Monnot, V.T. Paschos, and S. Toulouse. *Approximation polynomiale des problèmes NP-difficiles, Optima locaux et rapport différentiel.* Hermès, 2003.
84. J. Monnot, V.Th. Paschos, and S. Toulouse. Local approximations for maximum partial subgraph problem. *Operations Research Letters*, 2003. to appear.
85. J.B. Orlin, A.P. Punnen, and A. Schulz. Approximate local search in combinatorial optimization. In *SODA*, pages 587–596, 2004.
86. C.H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 163–169, 1991.
87. C.H. Papadimitriou. The complexity of the Lin-Kernighan heuristic for the traveling salesman problem. *SIAM Journal on Computing*, 21:450–465, 1992.
88. C.H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.
89. P.M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem : A survey and recent developments. In *DIMACS (Series in Discrete Mathematics and Theoretical Computer Science) Workshop*, volume 16, pages 1–42, May 20–21 1993.
90. S. Poljak. Integer linear programs and local search for max-cut. *SIAM Journal on Computing*, 24(4):822–839, 1995.

91. R.L. Rardin and M. Sudit. Paroids: a canonical format for combinatorial optimization. *Discrete Applied Mathematics*, 39:37–56, 1992.
92. R.L. Rardin and M. Sudit. Paroid search: generic local combinatorial optimization. *Discrete Applied Mathematics*, 43:155–174, 1993.
93. C.C. Ribeiro and P. Hansen. *Essays and surveys in metaheuristics.* Kluwer Academic Publishers, 2002.
94. A.A. Schäffer and M. Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991.
95. P. Schuurman and T. Vredeveld. Performance guarantee of local search for multiprocessor scheduling. In *IPCO*, LNCS 2081, pages 370–382. Springer, 2001.
96. S. Shimozono. Finding optimal subgraphs by local search. *Theoretical Computer Science*, 172:265–271, 1997.
97. D.B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.
98. C.A. Tovey. Local improvement on discrete structures. In E. Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 57–89. John Wiley and Sons, 1997.
99. L. Trevisan. Positive linear programming, parallel approximation and PCPs. In *4th Annual European Symposium on Algorithms, ESA '96*, LNCS 1136, pages 62–75. Springer, 1996.
100. V. Vazirani. *Approximation algorithms*. Springer, 2001.
101. T. Vredeveld and J.K Lenstra. On local search for the generalized graph coloring problem. *Operations Research Letters*, 31(1):28–34, 2003.
102. Y. Wan, G. Chen, and Y. Xu. A note on the minimum label spanning tree. *Information Processing Letters*, 84:99–101, 2002.
103. M. Yannakakis. The analysis of local search problems and their heuristics. In *STACS*, LNCS 415, pages 298–311. Springer, 1990.
104. M. Yannakakis. Computational complexity. In E. Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 19–55. John Wiley and Sons, 1997.

Approximation Algorithms for Path Coloring in Trees*

Ioannis Caragiannis[1], Christos Kaklamanis[1], and Giuseppe Persiano[2]

[1] Research Academic Computer Technology Institute and
Department of Computer Engineering and Informatics,
University of Patras, 26500 Rio, Greece
{caragian, kakl}@ceid.upatras.gr

[2] Dipartimento di Informatica ed Applicazioni,
Università di Salerno, 84081 Baronissi, Italy
giuper@dia.unisa.it

Abstract. The study of the path coloring problem is motivated by the allocation of optical bandwidth to communication requests in all-optical networks that utilize Wavelength Division Multiplexing (WDM). WDM technology establishes communication between pairs of network nodes by establishing transmitter-receiver paths and assigning wavelengths to each path so that no two paths going through the same fiber link use the same wavelength. Optical bandwidth is the number of distinct wavelengths. Since state-of-the-art technology allows for a limited number of wavelengths, the engineering problem to be solved is to establish communication minimizing the total number of wavelengths used. This is known as the *wavelength routing problem*. In the case where the underlying network is a tree, it is equivalent to the path coloring problem.

We survey recent advances on the path coloring problem in both undirected and bidirected trees. We present hardness results and lower bounds for the general problem covering also the special case of sets of symmetric paths (corresponding to the important case of symmetric communication). We give an overview of the main ideas of deterministic greedy algorithms and point out their limitations. For bidirected trees, we present recent results about the use of randomization for path coloring and outline approximation algorithms that find path colorings by exploiting fractional path colorings. Also, we discuss upper and lower bounds on the performance of on-line algorithms.

1 Introduction

1.1 Motivation

Optical fiber has been established as the standard transmission medium for backbone communication networks, since it can provide the required data rate, error rate and delay performance necessary for high speed networks of next

* This work was partially supported by the European Union under IST FET Project CRESCCO.

generation [19,36]. Multiwavelength communication [19,36] is the most popular communication technology used on optical networks. Roughly speaking, it allows to send different streams of data on different wavelengths along an optical fiber. Multiwavelength communication is implemented through *Wavelength Division Multiplexing* (WDM).

In a WDM all-optical network, once the data stream has been transmitted in the form of light, it continues without conversion to electronic form until it reaches its destination. WDM takes all data streams travelling on an incoming link and routes each of them to the right outgoing link, provided that each data stream travels on the same wavelength on both links. For a packet transmission to occur, a transmitter at the source must be tuned to the same wavelength as the receiver at the destination for the duration of the packet transmission and no data stream collision may occur at any fiber.

A WDM all-optical network can thus be modelled as a graph, where nodes of the graph are the nodes of the network and edges are optical fibers connecting nodes. Communication requests are ordered pairs of nodes to be thought of as transmitter-receiver pairs. The problem of *wavelength routing* consists of finding, for each transmitter-receiver pair, a path connecting the transmitter with the receiver and assigning a wavelength to each path, so that no two paths going through the same edge (fiber link) use the same wavelength. Intuitively, we may think of the wavelengths as colors and the wavelength assignment to paths as coloring. So, the wavelength routing problem consists of two subproblems: a *routing* problem (for selecting the path for each transmitter-receiver pair) and a *wavelength assignment* or *path coloring* problem (for coloring the paths established after solving the routing subproblem).

The objective is to minimize the *optical bandwidth*, that is the number of different wavelengths (colors) that are used. Optical bandwidth is a scarce resource. State-of-the-art technology allows some hundreds of wavelengths per fiber in the laboratory, even less in manufacturing, and there is no anticipation for dramatic progress in the near future. An efficient allocation of the optical bandwidth (with respect to the number of wavelengths used) is an important engineering problem that can have a significant impact on the success of the technology.

Theoretical work on optical networks mainly focuses on the performance of wavelength routing algorithms on regular networks using oblivious (predefined) routing schemes for connecting each transmitter to the corresponding receiver. We point out the pioneering work of Pankaj [33] who considered shuffle exchange, De Bruijn, and hypercubic networks. Aggarwal *et al.* [1] consider oblivious routing schemes and wavelength assignment algorithms in several networks. Raghavan and Upfal in [35] consider mesh-like networks. Aumann and Rabani [3] improve the bounds of Raghavan and Upfal for mesh networks and also give tight results for hypercubic networks. Rabani in [34] gives almost optimal results for the wavelength routing problem on meshes and mesh-like networks improving earlier results implicit in [26].

These topologies reflect architectures of optical computers rather than wide-area networks. For fundamental practical reasons, the telecommunication industry does not deploy massive regular architectures: backbone networks need to reflect irregularity of geography, non-uniform clustering of users and traffic, hierarchy of services, dynamic growth, etc. In this direction, Raghavan and Upfal [35], Aumann and Rabani [3], and Bermond *et al.* [7], among other results, focus on bounded-degree networks and give upper and lower bounds as functions of the network expansion.

However, wide-area multiwavelength technology is expected to grow around the evolution of current networking principles and existing fiber networks. These are mainly SONET (Synchronous Optical Networking Technology) rings and trees [31]. In this sense, even asymptotic results for expander graphs do not address the above telecommunication scenario. In this direction, Kumar [27] studies the wavelength routing problem in rings improving previous earlier results implicit in [38,24].

Most of the works mentioned above model the optical network as an undirected graph. Indeed, an optical fiber connecting two nodes can be used to carry traffic in both directions. However, additional devices which are placed on fibers like amplifiers work on only one direction and two “opposite directed” fibers are used in practice to support bidirectional communication between pairs of nodes. So, it is reasonable to model optical networks as bidirected graphs, i.e., graphs whose nodes are connected through pairs of opposite directed edges.

In this work, we consider tree topologies and survey recent methods and algorithms for efficiently coloring paths on undirected and bidirected trees. Notice that, in a tree, the path connecting two nodes is well-defined; this means that the wavelength routing in trees is equivalent to path coloring. We consider arbitrary sets of paths. Surveys on the path coloring problem for sets of paths which capture specific communication patterns like *broadcasting*, *gossiping*, and *permutation routing* can be found in [5,25].

1.2 Problem Definition

We now formulate the path coloring problem in detail by giving the necessary definitions. A path p on a tree T is a sequence $p = (u_1, \cdots, u_l)$ of nodes of the tree such that (u_i, u_{i+1}) is an edge of the tree for $i = 1, \cdots, l-1$. We say that a path touches a node if the node belongs to the sequence of nodes forming the path. Nodes u_1 and u_l are called the origin and the destination of the path, respectively. We also say that path p includes edges (u_i, u_{i+1}) for $i = 1, \cdots, l-1$. The tree can be undirected or bidirected. Respectively, the path can be undirected or directed; in the latter case, a path consists of directed edges of the tree.

Given a set of undirected (resp. directed) paths $P = \{p_1, \cdots, p_k\}$ on a undirected (resp. bidirected) tree, we define the load of an undirected (resp. directed) edge e as the number of paths of P that include e. The *load* of the set of paths P is defined as the maximum edge load over all edges of the tree. The decision version of the path coloring problem can be formalized as follows.

Path Coloring in Trees
Instance: An undirected (resp. bidirected) tree T, a set of undirected (resp. directed) paths P on T and an integer B.
Question: Is there a coloring $\chi : P \rightarrow \{1, \cdots, B\}$ such that, for any two paths p_1 and p_2 of P, if p_1 and p_2 share an undirected (resp. directed) edge of T, then $\chi(p_1) \neq \chi(p_2)$?

In the optimization version of the path coloring problem, we are given a tree T and a set of paths P on T, and the goal is to compute the minimum integer B such that the answer to the corresponding instance (T, P, B) of the decision problem is YES, i.e., the goal is to compute the minimum number of colors sufficient for coloring P and, usually, to provide the corresponding coloring. Obviously, if $B < L$, the answer to the instance (T, P, B) of the decision problem is certainly NO. Therefore, the load of a set of paths P is a lower bound on the minimum number of colors sufficient for coloring P.

Path coloring is similar to graph coloring. Given an undirected graph G, the graph coloring problem is to assign colors to the nodes of G so that any two adjacent nodes are colored with different colors. Note that path coloring of a set of paths P on a tree T is equivalent to graph coloring of the *conflict graph* of P, i.e., the graph which contains one node for each path in P and an edge between two nodes if the corresponding paths share an edge of the tree. In the text below, we also refer to the edge-coloring problem; here, we seek for assignment of colors to the edges of an undirected graph so that any two edges incident to the same node are assigned different colors.

1.3 Roadmap

The rest of this paper is structured as follows. In Section 2, we present hardness results and lower bounds for both the undirected and the directed version of the problem. We also address the special case of symmetric sets of paths (corresponding to an important communication pattern in today's optical networks) in bidirected trees. In Section 3, we describe deterministic greedy algorithms for path coloring in both undirected and bidirected trees and present the best known results for them. Especially for path coloring in bidirected trees, in Section 4 we discuss randomized greedy algorithms, while we present algorithms that exploit fractional path colorings to compute approximate path colorings in Section 5. In Section 6 we present algorithms and lower bounds for the on-line version of the problem. We conclude, in Section 7, with a list of open problems.

2 Hardness Results and Lower Bounds

As we mentioned in the previous section, the load of a set of paths is a lower bound on the number of necessary colors. We now present better lower bounds for both directed and undirected sets of paths. It is easy to construct a set of undirected paths on an undirected binary tree which requires at least $3L/2$ colors (see Figure 1a). The next result shows that the number of colors can be sensibly higher than the load in the case of directed path coloring as well.

Theorem 1 (Kumar and Schwabe [29]). *For any integer $l > 0$, there exists a set of directed paths of load $L = 4l$ on a binary bidirected tree T that requires at least $5L/4$ colors.*

For proving the theorem, a set of $5L/2$ directed paths is constructed on a binary bidirected tree with six nodes. The set of paths is such that no more than two paths can be colored with the same color. This implies that at least $5L/4$ colors are necessary. The construction is depicted in Figure 1b.

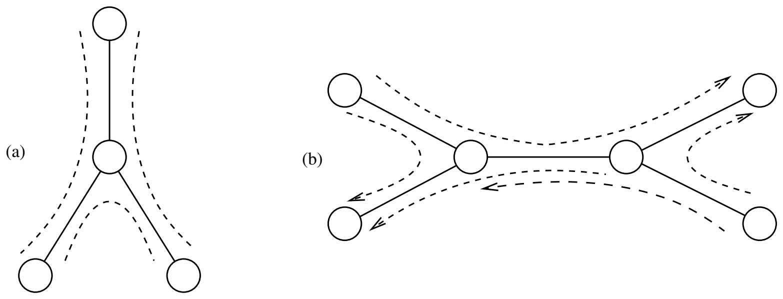

Fig. 1. (a) A set of undirected paths of load L that requires $3L/2$ colors. Each dashed line represents $L/2$ parallel undirected paths. (b) A set of directed paths of load L that requires $5L/4$ colors. Each arrow represents $L/2 = 2l$ parallel directed paths.

The next three theorems show that, in general, computing the optimal number of colors is NP-hard. Erlebach and Jansen [14] and, independently, Kumar *et al.* [28] have proved the following hardness results.

Theorem 2 (Erlebach and Jansen [14], Kumar *et al.* [28]). *Given an undirected star T and a set of paths P of load $L = 3$ on T, it is NP-complete to decide whether P can be colored with 3 colors.*

Theorem 3 (Erlebach and Jansen [14], Kumar *et al.* [28]). *Given a bidirected tree T of depth at least 2 and a set of paths P of load $L = 3$ on T, it is NP-complete to decide whether P can be colored with 3 colors.*

The proofs of the above theorems use reductions from the edge coloring problem in graphs of maximum degree 3 [21]. Actually, as it was first proved in [18], the path coloring problem in undirected stars is equivalent to edge coloring of multigraphs.

Note that the above statement holds in bidirected trees of arbitrary maximum degree. The following result applies to binary bidirected trees and sets of directed paths of arbitrary load. The proof uses a reduction from the circular arc coloring problem [16].

Theorem 4 (Erlebach and Jansen [14], Kumar *et al.* [28]). *Given a bidirected binary tree T and a set of paths P of load L on T, it is NP-complete to decide whether P can be colored with L colors.*

Thus, the corresponding optimization problems (minimizing the number of colors) are NP-hard, in general. On the positive side, a few special cases of the problem can be solved in polynomial time either by using dynamic programming or by exhaustively searching many different solutions.

Theorem 5 (Erlebach and Jansen [14], Kumar *et al.* [28]). *The path coloring problem can be solved optimally in undirected trees of bounded degree.*

Theorem 6 (Erlebach and Jansen [14]). *Given a set of (undirected or directed) paths P on a (undirected or bidirected) tree T with n nodes, such that at most $O\left(\frac{\log n}{\log\log n}\right)$ paths touch any node of T, an optimal coloring of P can be computed in polynomial time.*

As a corollary, we obtain that optimally coloring sets of paths of load $O\left(\frac{\log n}{\log\log n}\right)$ in bidirected trees of bounded degree can be done in polynomial time.

2.1 Coloring Symmetric Sets of Paths

We now consider the special case of sets of symmetric paths in bidirected trees. Two directed paths are called symmetric if the origin of the one is the destination of the other and vice versa. A set of directed paths is called symmetric if it can be partitioned into pairs of symmetric paths.

We can restrict the coloring of symmetric sets of paths to use the same color to color the two paths in each pair of symmetric paths. Then, we can solve the path coloring problem using any algorithm for the undirected version of the problem. In this way, up to $3L/2$ colors may be necessary (by the discussion at the beginning of Section 2) and sufficient (by using an algorithm presented in Section 3.1, see Theorem 9). The following two questions now arise.

– Can we improve this bound if we do not constrain symmetric paths to use the same color?
– Is the path coloring problem easier than in the general case if we restrict the input instances to sets of symmetric paths?

Although we give a positive answer for the first question in the case of binary bidirected trees in Section 4, we are not aware of complete answers for these questions. The following discussion shows some inherent similarities and differences between the general path coloring problem and the case where the input instance is restricted to sets of symmetric paths.

For these sets of paths, Caragiannis *et al.* [10] have proved some interesting statements (lower bounds). Both NP-completeness results (Theorems 3 and 4) hold in the case of sets of symmetric paths [10]. The proofs again use slightly modified reductions from the edge coloring and circular arc coloring problem. Notice that the lower bound of Figure 1 applies to non-symmetric sets of paths. A similar lower bound (but much more complicated than the one of Figure 1b) holds for sets of symmetric paths as well.

Theorem 7 (Caragiannis *et al.* [10]). *For any $\delta > 0$, there exists an integer $l > 0$, a binary bidirected tree T and a set of symmetric paths P with load $L = 4l$ on T, such that no algorithm can color P using less than $(5/4 - \delta)L$ colors.*

Does Theorem 7 indicate that the symmetric version of the problem is as "hard" as the general one? The following result gives evidence for a negative answer. Notice that for sets of symmetric paths, there exists an algorithm which may color paths in such a way that each pair of opposite directed edges sees at most L colors. This can be done in a trivial way if we consider two symmetric paths as an undirected one and use any algorithm for the undirected problem. However, this is not the case for the non-symmetric problem.

Theorem 8 (Caragiannis *et al.* [10]). *For any integer $l > 0$, there exists a bidirected tree T and a set of directed paths with load $L = 8l$ that cannot be colored in such a way that any pair of opposite directed edges of T sees less than $9L/8$ colors.*

3 Greedy Algorithms

Most of the known path coloring algorithms [12,14,17,22,29,35] belong to a special class of algorithms, the class of *greedy* algorithms. We devote this section to their study. Given a tree T and a set of paths P, we call greedy a path coloring algorithm that works as follows:

> Starting from an arbitrary node (the root of the tree), the algorithm computes a breadth-first (BFS) numbering of the nodes of the tree.
> The algorithm proceeds in phases, one per each node u of the tree. The nodes are considered following their BFS numbering. The phase associated with node u takes as input a proper coloring of all the paths that touch nodes with BFS number strictly smaller than u's; all other paths have not been colored yet.
> During the phase associated with node u, the partial proper coloring is extended to a proper one that assigns colors to paths that touch node u but have not been colored yet.
> During each phase, the algorithm does not recolor paths that have been colored in previous phases.

The various greedy algorithms differ among themselves with respect to the strategies followed to extend the partial proper coloring during a phase. In the following two sections, we describe the techniques used by greedy algorithms in undirected and bidirected trees. Then, we describe the main technique used for proving lower bounds for greedy path coloring algorithms in bidirected trees.

3.1 Greedy Path Coloring Algorithms in Undirected Trees

The algorithms in [14,35] reduce the coloring of a phase to an edge coloring problem on a multigraph. In the following we describe this reduction.

Consider a set of paths P of load L on an undirected tree T and the phase of a greedy algorithm associated with a node u of T. The phase receives as input the coloring of the paths of P in previous phases (i.e., in phases associated with nodes of T with numbers smaller than u's) and extends this coloring by coloring the paths touching node u which are still uncolored by solving an edge-coloring problem on a multigraph G_u. The multigraph G_u associated with node u is constructed as follows. We denote by v_0 the parent of u and let $v_1, ..., v_k$ be the children of u. For each node v_i of the nodes $v_0, v_1, ..., v_k$, the graph G_u has two nodes x_i and y_i. For each path originated from u and going to v_i, we add an edge between x_i and y_i in G_u. For each path between two neighbors v_i and v_j of u, we add an edge between nodes x_i and x_j in G_u. It can be easily seen that the graph G_u has maximum degree L. An example of the construction is depicted in Figure 2.

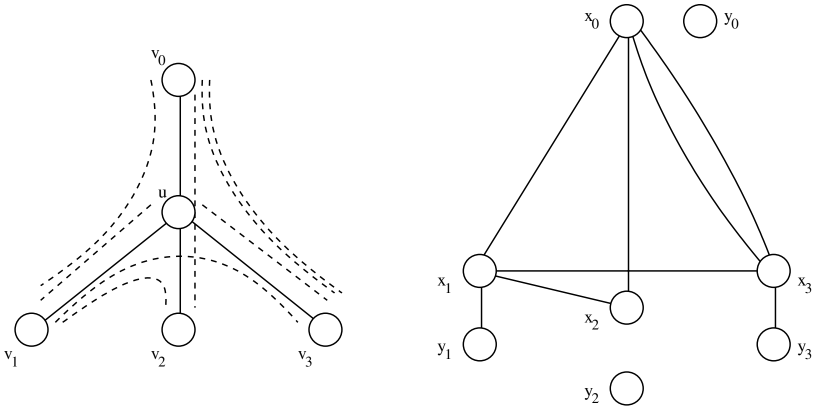

Fig. 2. Undirected paths touching node u and the corresponding multigraph

Intuitively, edges of G_u incident to a node x_i correspond to paths of P traversing the edge of T between nodes u and v_i. The edges of G_u incident to node x_0 correspond to paths of P traversing the edge of T between node u and its parent node v_0; these paths have been colored in phases associated with nodes having BFS numbers smaller than u's. We call the edges of G_u incident to node x_0 *color-forced* edges. The phase associated with node u colors the edges of G_u under the constraint that color-forced edges are assigned the colors assigned to the corresponding paths in previous phases. It can be easily seen that these constraints do not make the edge-coloring problem harder with respect to the number of colors required since, in any proper coloring of the edges of G_u (without constraints on the coloring of the edges incident to x_0), different colors must be assigned to the color-forced edges. These constraints just reduce the number of different proper colorings. The produced coloring of the edges of G_u trivially yields a coloring of the paths of P touching node u and nodes with BFS numbers smaller than u's. Also, note that the optimal number of colors required for coloring the edges of G_u is a lower bound on the number of colors required for coloring P.

For extending the coloring to all edges of G_u, colors used for coloring the edges of multigraphs associated with nodes with numbers smaller than u's which are not used in the edges incident to node x_0 of G_u can be used. New colors are used only when the colors already used do not suffice to complete the edge-coloring. So, the total number of colors used for coloring the paths of P is the maximum over all phases of the number of colors used for coloring the edges of the corresponding multigraph. Raghavan and Upfal [35] used an algorithm of Shannon [37] which colors the edges of a multigraph with maximum degree L using at most $3L/2$ colors to obtain the following result.

Theorem 9 (Raghavan and Upfal [35]). *There exists a polynomial-time algorithm which colors any set of paths of load L on an undirected tree using at most* $3L/2$ *colors.*

From the discussion on lower bounds in Section 2, we know that this bound is tight. Erlebach and Jansen [14] observed that using an algorithm which colors the edges of a multigraph G with at most $f(\chi(G))$ colors, where f is a non-decreasing function and $\chi(G)$ denotes the optimal number of colors for coloring the edges of G, we obtain a greedy path coloring algorithm for undirected trees which colors any set of paths P using $f(\chi(P))$ colors, where $\chi(P)$ denotes the optimal number of colors sufficient for coloring P. This can be explained as follows. Let u be the node for which the edge-coloring algorithm uses the maximum number of colors for coloring the edges of the corresponding multigraph G_u. Clearly, $\chi(P) \geq \chi(G_u)$, while the greedy algorithm uses at most $f(\chi(G_u)) \geq f(\chi(P))$ colors to color the paths of P. In [32], Nishizeki and Kashiwagi present an algorithm for coloring the edges of a multigraph G with at most $\lfloor 1.1\chi(G) + 0.8 \rfloor$ colors. Combined with the discussion above, this implies the following.

Theorem 10 (Erlebach and Jansen [14]). *There exists a polynomial-time algorithm which colors any set of paths P on an undirected tree using at most* $\lfloor 1.1\chi(P) + 0.8 \rfloor$ *colors, where* $\chi(P)$ *denotes the optimal number of colors sufficient for coloring P.*

Actually, Theorem 5 and Theorem 6 (for undirected trees) can be proved using greedy path coloring algorithms. In both cases, an edge-coloring algorithm is used in each phase to optimally color the edges of the corresponding multigraph in polynomial time. In the case of Theorem 5, the multigraph associated with each phase has a constant number of nodes; in this case, the edge-coloring problem can be solved in time polynomial in L. In the case of Theorem 6, the multigraph associated with each phase has at most $O\left(\frac{\log n}{\log \log n}\right)$ edges; it can be easily seen that an optimal edge coloring can be computed in time polynomial in n.

3.2 Greedy Path Coloring Algorithms in Bidirected Trees

The algorithms in [15,29] for path coloring in bidirected trees reduce the coloring of a phase to an edge-coloring problem on a bipartite multigraph. In the following we describe this reduction.

Consider a set of directed paths P of load L on a bidirected tree T and the phase of a greedy algorithm associated with a node u of T. The phase receives as input the coloring of the paths of P in previous phases (i.e., in phases associated with nodes of T with numbers smaller than u's) and extends this coloring by coloring the paths touching node u which are still uncolored by solving an edge-coloring problem on a bipartite multigraph H_u. The multigraph H_u associated with node u is constructed as follows. Again, we denote by v_0 the parent of u and let $v_1, \cdots, v_k$ be the children of u. For each node v_i, the bipartite multigraph has four nodes w_i, x_i, y_i, z_i and the left and right partitions are $\{w_i, x_i | i = 0, \cdots k\}$ and $\{z_i, y_i | i = 0, \cdots k\}$, respectively. For each path of the tree directed out of some node v_i into some node v_j, the bipartite multigraph H_u has an edge between w_i and z_j. For each path directed out of some node v_i and terminating at u, we have an edge between w_i and y_i. Finally, for each path directed out of u into some node v_i, we have an edge between z_i and x_i. It can be easily seen that the graph H_u has maximum degree L. An example of this construction is depicted in Figure 3.

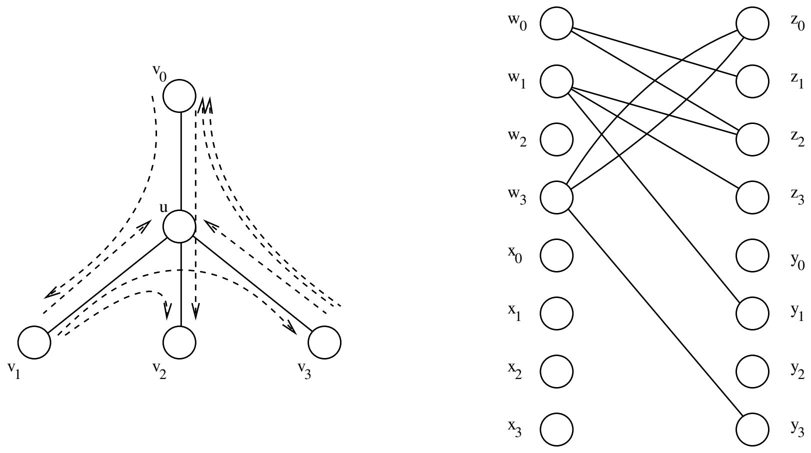

Fig. 3. Directed paths touching node u and the corresponding bipartite multigraph

Intuitively, edges of H_u incident to a node w_i correspond to paths of P traversing the directed edge of T from node v_i to node u, while edges of H_u incident to a node z_i correspond to paths of P traversing the directed edge of T from node u to node v_i. Observe that no edges extend across opposite nodes w_i and z_i as they would correspond to paths starting and ending on the same node. The edges of H_u incident to nodes w_0 and z_0 correspond to paths of P traversing the opposite directed edges between u and its parent node v_0; these paths have been colored in phases associated with nodes having BFS numbers smaller than u's. We call the edges of H_u incident to nodes w_0 and z_0 color-forced edges. The phase associated with node u colors the edges of H_u under the constraint that color-forced edges are assigned the colors assigned to the corresponding directed paths in previous phases. The produced coloring of the edges of H_u trivially yields a coloring of the paths of P touching node u and nodes with BFS numbers smaller than u's. Similarly to the case of path coloring in undirected trees, the total number of colors used for coloring the paths of P can be made as small as the maximum over all phases of the number of colors used for coloring the edges of the corresponding bipartite multigraph.

Thus, the problem of coloring paths at each phase is reduced to the problem of coloring the edges of a bipartite multigraph of maximum degree L, under the constraint that some colors have already been assigned to the color-forced edges. We call this problem an *α-constrained bipartite edge coloring problem* on a bipartite graph with maximum degree L. The parameter α denotes that color-forced edges have been colored with at most αL colors ($1 \leq \alpha \leq 2$). Note that unlike the case of edge-coloring multigraphs produced by undirected sets of paths, the existence of constraints in the color-forced edges of the bipartite multigraph H_u associated with node u indeed make the problem harder with respect to the total number of colors required for coloring all the edges. Bipartite multigraphs with maximum degree L can always be colored with L colors (if no constraints on the color-forced edges exist). However, the bipartite multigraphs at the phases of a greedy algorithm for coloring directed paths on bidirected trees may require at least $5L/4$ colors due to the lower bound presented in Section 2 (Theorem 1).

In the following, we first describe a simple solution to the 2-constrained bipartite edge coloring. In particular we show how to color the edges of a bipartite multigraph H_u of maximum degree L in which the color-forced edges are constrained to be colored with at most $2L$ specific colors using at most $2L$ colors in total. Then, we briefly describe the main ideas that lead to an improved solution of the problem in [15].

We call *single* colors the colors that appear only in one color-forced edge and *double* colors the colors that appear in two color-forced edges (one incident to w_0 and one incident to z_0). We denote by S and D the number of single and double colors, respectively. Clearly, $2D + S \leq 2L$.

We decompose the bipartite graph into L matchings (this can be done in polynomial time since the graph is bipartite and has maximum degree L; see [6]). At least $S/2$ of these matchings have at least one single color in one of the two colored-forced edges. Thus, we can use this single color to color the uncolored edges of the matching. The uncolored edges of the matchings that have no color-forced edges colored with a single color are colored with extra colors (one extra color per matching). In total, we use at most

$$D + S + L - S/2 = L + D + S/2 \leq 2L$$

colors.

In each phase of the greedy algorithm when applied to a set of directed paths of load L on a bidirected tree, an instance of 2-constrained bipartite edge coloring on a bipartite multigraph of maximum degree L is solved. The number of colors used in each phase never grows larger than $2L$. In this way, we obtain a simple greedy path coloring algorithm that uses at most $2L$ colors for any set of directed paths of load L on a bidirected tree.

Better solutions are obtained in [15,29]. Again consider the application of the greedy algorithm on a set of directed paths of load L on a bidirected tree and let u be a node with parent v_0 and children $v_1, ..., v_k$. For $i = 0, ..., k$, we call the pair of nodes w_i, z_i of the bipartite multigraph H_u a *line*. We say that the color c is used in the line w_i, z_i if the color c is used in some edge incident either to node w_i or to node z_i. Erlebach *et al.* [15] solve the problem proving the following theorem (a slightly inferior result was obtained in [29]).

Theorem 11 (Erlebach *et al.* [15]). *There exists a polynomial time algorithm which, for any $\alpha \in [1, 4/3]$, colors an instance of the α-constrained bipartite edge coloring problem on the bipartite multigraph H_u of maximum degree L using at most $\left(1 + \frac{\alpha}{2}\right) L$ total colors so that the number of colors used per each line of H_u is at most $4L/3$.*

Note that the edges of H_u incident to a line w_i, z_i correspond to directed paths traversing the opposite directed edges of the tree between u and a child v_i of u. This means that, if we use the edge-coloring algorithm of [15] in the phase associated with node u for solving an instance of the α-constrained bipartite edge-coloring on the bipartite multigraph H_u of maximum degree L with $\alpha \leq 4/3$, then, the edge-coloring problems that have to be solved in the phases associated with the child nodes of u are instances of the α-constrained bipartite edge coloring on bipartite multigraph of maximum degree L for $\alpha \leq 4/3$ as well. In the first phase of the greedy algorithm (i.e., the one associated with the root of the tree), there are no constraints on the edges of the corresponding bipartite multigraph, so its edges can be properly colored with at most L colors. Then, we can easily verify inductively that, at each of the next phases, the number of colors used in the paths colored in previous phases is never more than $4L/3$. So, in all phases of the greedy algorithm, an instance of the α-constrained bipartite edge coloring problem with $\alpha \leq 4/3$ has to be solved in a bipartite multigraph of maximum degree L and the total number of colors used is never more than $5L/3$. This is summarized in the following theorem.

Theorem 12 (Erlebach *et al.* [15]). *There exists a polynomial time greedy algorithm which colors any set of directed paths of load L on a bidirected tree using at most $5L/3$ colors.*

The interested reader may refer to the papers [15,29] for detailed description of the edge-coloring techniques. They either consider matchings in pairs and color them in sophisticated ways using detailed potential and averaging arguments for the analysis [29] or partition matchings into groups which can be colored and accounted for independently [15]. Note that one might think of bipartite edge coloring problems with different constraints. Tight bounds on the number of colors for more generalized constrained bipartite edge coloring problems can be found in [11].

The algorithms in [12,22] use much simpler methods to obtain the same upper bound in bidirected binary trees. The common characteristic of all the algorithms discussed so far is that they are deterministic.

3.3 A Lower Bound Technique

In this section, we present a lower bound technique for greedy path coloring algorithms in bidirected trees. We first briefly describe the technique; then we give the best known statement for deterministic greedy algorithms.

The technique is based on an adversary argument. An adversary algorithm $\mathcal{ADV}$ is exhibited that constructs a set of directed paths on a binary bidirected tree for which it can be proved that there exists a lower bound on the number of colors used by any greedy algorithm.

The adversary $\mathcal{ADV}$ constructs the set of directed paths P in an incremental way visiting the nodes of the tree according to a BFS order. At each node u, the $\mathcal{ADV}$ deals with the set of paths traversing one of the two parallel directed edges between u and its parent $p(u)$. For each *downward* path p (that is, for each path including the directed edge $(p(u), u)$), $\mathcal{ADV}$ has two options:

1. do nothing; i.e., make p stop at u;
2. propagate p to the left child $l(u)$ by appending edge $(u, l(u))$ to p.

Similarly, for each *upward* path p (that is, for each path including the directed edge $(u, p(u))$), $\mathcal{ADV}$ has two options:

1. do nothing; i.e. make p start from u;
2. make p originate from the right child $r(u)$ by pre-pending edge $(r(u), u)$ to p;

Moreover, the adversary algorithm $\mathcal{ADV}$ can introduce directed paths between the two children of u (see Figure 4). Initially, these paths will consist of only two edges (from a child to u and from u to the other child) and can be augmented when the adversary reaches the children of u.

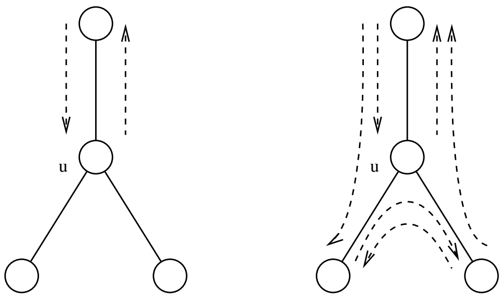

Fig. 4. The construction for the lower bound for greedy path coloring algorithms in bidirected trees

At each step, since the adversary $\mathcal{ADV}$ may know how a greedy algorithm performs the coloring, it can choose to augment paths in such a way that the number of common colors used in downward paths from $p(u)$ to $l(u)$ and upward paths from $r(u)$ to $p(u)$ is small. In this way, no matter what a greedy algorithm can do, both the total number of colors and the number of colors seen by the opposite directed edges between u and its children will increase.

By constructing such an adversary for deterministic greedy algorithms, Erlebach *et al.* [15] prove the following lower bound. The same lower bound technique can be used for greedy algorithms that use randomization (see Section 4).

Theorem 13 (Erlebach *et al.* [15]). *Let $\mathcal{A}$ be a deterministic greedy path coloring algorithm in bidirected trees. There exists an algorithm $\mathcal{ADV}$ which, on input $\delta > 0$ and integer $L > 0$, outputs a binary bidirected tree T and a set of directed paths P of load L on T, such that $\mathcal{A}$ colors P with at least $(5/3 - \delta)L$ colors.*

Thus, the greedy algorithm presented in [15] (see also Theorem 12) is the best possible within the class of deterministic greedy algorithms for path coloring in bidirected trees. In Section 4 we demonstrate how randomization can be used to beat the 5/3 barrier at least on binary bidirected trees.

4 Randomized Algorithms in Bidirected Trees

As we discussed in Section 3.1, the algorithm of Raghavan and Upfal [35] achieves a tight upper bound on the number of colors sufficient for path coloring in undirected trees while the use of efficient approximation algorithms for edge-coloring of multigraphs leads to very good approximations for the problem. This is not the case for the directed version of the problem. In this section, we discuss how randomization can be used in order to improve the upper bounds for path coloring in binary bidirected trees.

In an attempt to beat the 5/3 lower bound for deterministic greedy algorithms for path coloring in bidirected trees, Auletta *et al.* [2] define the class of randomized greedy path coloring algorithms. Randomized greedy algorithms have the same structure as deterministic ones; that is, starting from a node, they consider the nodes of the tree in a BFS manner. Their main difference is that a randomized greedy algorithm $\mathcal{A}$ uses a palette of colors and at each phase associated with a node, $\mathcal{A}$ picks a random proper coloring of the uncolored paths using colors of the palette according to some probability distribution.

The results presented in the following were originally obtained in [2]. The interested reader may see [2] for further details.

4.1 Lower Bounds

In this section, we present two lower bounds on the number of colors used by randomized greedy algorithms to color sets of paths of load L on bidirected trees. We first present a lower bound for large trees (i.e., trees of depth $\Omega(L)$) in

Theorem 14; then, in Theorem 15, we present a lower bound for trees of constant depth.

The first lower bound states that no randomized greedy algorithm can achieve a performance ratio better than 3/2 if the depth of the tree is large.

Theorem 14 (Auletta *et al.* [2]). *Let $\mathcal{A}$ be a (possibly randomized) greedy path coloring algorithm on binary bidirected trees. There exists a randomized algorithm $\mathcal{ADV}$ which, on input $\epsilon > 0$ and integer $L > 0$, outputs a binary bidirected tree T of depth $L + \epsilon \ln L + 2$ and a set of directed paths P of load L on T, such that the probability that $\mathcal{A}$ colors P with at least $3L/2$ colors is at least $1 - \exp(-L^{\epsilon})$.*

The following lower bound holds even for trees of constant depth.

Theorem 15 (Auletta *et al.* [2]). *Let $\mathcal{A}$ be a (possibly randomized) greedy path coloring algorithm on binary bidirected trees. There exists a randomized algorithm $\mathcal{ADV}$ which, on input $\delta > 0$ and integer $L > 0$, outputs a binary bidirected tree T of constant depth and a set of directed paths P with load L on T, such that the probability that $\mathcal{A}$ colors P with at least $(1.293 - \delta - o(1))L$ colors is at least $1 - O(L^{-2})$.*

For the proofs, constructions based on the lower bound technique presented in Section 3.3 are used.

Note that the adversary assumed in Theorems 14 and 15 has no knowledge of the probability distribution according to which the randomized greedy algorithm makes its random choices. Possibly, better lower bounds may be achieved by considering more powerful adversaries.

4.2 Upper Bounds

In this section, we give the main ideas of a randomized path coloring algorithm presented in [2]. The algorithm has a greedy structure but allows for limited recoloring at the phases associated with each node.

At each phase, the path coloring algorithm maintains the following two invariants:

I. The total number of colors is no greater than $7L/5$.
II. The number of colors seen by two opposite directed edges is exactly $6L/5$.

At a phase associated with a node u, a *coloring procedure* is executed which extends the coloring of paths that touch u and its parent node to the paths that touch u and are still uncolored. The coloring procedure is randomized. It selects the coloring of paths being uncolored according to a specific probability distribution on random colorings in which the probability that two paths crossing an edge between u and any child of u in opposite directions are assigned the same color is $\frac{4}{5L}$. In this way, the algorithm can complete the coloring at the phase associated with node u using at most $7L/5$ colors in total, keeping the number of colors seen by the opposite directed edges between u and its children to $6L/5$.

At each phase associated with a node u, the algorithm is enhanced by a *recoloring procedure* which recolors a small subset of paths in order to maintain some specific properties on the (probability distribution of the) coloring of paths touching u and its parent. This procedure is randomized as well.

The recoloring procedure at each phase of the algorithm works with very high probability. The coloring procedure at each phase always works correctly maintaining the two invariants. As a result, if the depth of the tree is not very large (i.e., $o(L^{1/3})$), the algorithm executes the phases associated with all nodes, with high probability.

After the execution of all phases, the set of paths being recolored by the executions of the recoloring procedure are colored using the simple deterministic greedy algorithm with at most $o(L)$ extra colors due to the fact that as far as the depth of the tree is not very large (i.e., $o(L^{1/3})$), the load of the set of paths being recolored is at most $o(L)$, with high probability.

In this way, the following result is proved. The interested reader may look at [2] for a detailed description of the algorithm and formal proofs.

Theorem 16 (Auletta *et al.* [2]). *Let $0 < \delta < 1/3$ be a constant. There exists a randomized path coloring algorithm that colors any set of directed paths of load L on a binary bidirected tree of depth at most $L^{\delta}/8$ using at most $7L/5 + o(L)$ colors, with probability at least $1 - \exp\left(-\Omega(L^{\delta})\right)$.*

Going back to the discussion of Section 2.1 on symmetric sets of paths, Theorem 16 immediately implies that, at least for symmetric sets of paths on binary bidirected trees (with some additional restrictions on their depth), allowing symmetric paths to use different colors leads to better colorings (i.e., colorings with less than $3L/2$ colors). A better algorithm for symmetric sets of paths on binary trees has been presented in [13]. It colors any symmetric set of paths of load L on a binary bidirected tree with the same restrictions with those in Theorem 16 using at most $1.367L + o(L)$ colors, with high probability. This algorithm follows the main ideas behind the randomized algorithm discussed above.

5 Fractional Path Coloring and Applications

So far, we have expressed the performance of the path coloring algorithms in terms of the load of the set of paths to be colored. The approximation guarantee is easily achieved since the load is a lower bound on the number of colors necessary for coloring a set of paths.

In this section, among other results, we present approximation algorithms for which we can express their performance in terms of both the load and the *fractional path chromatic number* of the set of paths to be colored. In this way, we obtain a better approximation guarantee at least for bidirected trees of bounded degree.

We first give some definitions. The graph coloring problem can be considered as finding a minimum cost integral covering of the nodes of a graph by independent sets of unit cost. Given a graph $G = (V, E)$, this means solving the following integer linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{I\in\mathcal{I}} x(I) & \\ \text{subject to} & \sum_{I\in\mathcal{I}:v\in I} x(I) \geq 1 & \forall\, v \in V \\ & x(I) \in \{0,1\} & \forall\, I \in \mathcal{I} \end{array}$$

where $\mathcal{I}$ denotes the set of the independent sets of G.

This formulation has a natural relaxation into the following linear program:

$$\begin{array}{lll} \text{minimize} & \sum_{I\in\mathcal{I}} \bar{x}(I) & \\ \text{subject to} & \sum_{I\in\mathcal{I}:v\in I} \bar{x}(I) \geq 1 & \forall\, v \in V \\ & 0 \leq \bar{x}(I) \leq 1 & \forall\, I \in \mathcal{I} \end{array}$$

The corresponding combinatorial problem is called the *fractional coloring problem* (see [20]), and the value of an optimal solution is called the *fractional chromatic number*. Clearly, the fractional chromatic number of a graph is a lower bound on its chromatic number. If $\bar{x}$ is a function over the independent sets of the graph G satisfying the constraints of the linear program, we call it a *fractional coloring* of G.

We now extend some terms of graph coloring to path coloring. Given a set of directed paths P on a bidirected tree T, we define an *independent set of paths* as a set of pairwise edge-disjoint paths, i.e., a set of paths whose corresponding nodes form an independent set of the conflict graph. If G is the conflict graph of P on T, we will denote by $w_f(P)$ the fractional chromatic number of G and call it the *fractional path chromatic number* of P on T. Clearly, $w_f(P)$ is a lower bound on the optimal number of colors sufficient for coloring P. Also, a fractional coloring of the conflict graph G is called a *fractional path coloring* of P.

In general, the fractional chromatic number is as hard to approximate as the chromatic number. Indeed, the size of the above described linear program is exponential (proportional to the number of independent sets of G). However, in [8], it is shown how to compute an optimal fractional path coloring. In particular, the following statement is proved.

Theorem 17 (Caragiannis *et al.* [8]). *Fractional path coloring in bounded-degree trees can be computed in polynomial time.*

The main idea for the proof of Theorem 17 is to inductively construct a linear program whose solution is actually the optimal solution to the fractional path coloring problem. The linear program produced is proved to have polynomial size (i.e., polynomial number of variables) as far as the degree of the tree is bounded. In this way, fractional path coloring in trees is reduced to solving a polynomial size linear program.

5.1 Fractional Path Coloring on Binary Bidirected Trees

In this section we give an upper bound on the fractional path chromatic number of a set of directed paths on a binary bidirected tree. We express the result in terms of the load. This result was first discussed in [8].

In Section 4 we discussed the randomized algorithm presented in [2], which colors a set of directed paths of load L on a binary bidirected tree using at most $7L/5 + o(L)$ colors. The algorithm works with high probability under some additional constraints (i.e., the depth of the tree is not very large). The hidden constants in the low order term are large and are due to the integrality constraints of the problem and the random choices of the algorithm.

In the case of fractional coloring we can design a deterministic algorithm having the same top-down structure as the path coloring algorithm in [2]. Given a set of directed paths P of load L on a bidirected tree, this fractional path coloring algorithm uses fractions of colors to color paths and at each step, it maintains the following invariant: the common fractions of colors assigned to any two paths traversing an edge of the tree in opposite directions is $\frac{4}{5L}$. Following simpler analysis than in [2], we can obtain that, given a set of directed paths P on a binary bidirected tree T, the algorithm computes a fractional path coloring of P with cost at most $7L/5$. Comparing this result to the randomized algorithm presented in Section 4 and particularly to Theorem 16, we observe that the low order term is now eliminated. This is due to the fact that, when we compute a fractional coloring, we have no integrality constraints and the algorithm is deterministic.

Theorem 18 (Caragiannis *et al.* [8]). *For any set of directed paths of load L on a binary bidirected tree, there exists a fractional path coloring of cost at most $7L/5$. Moreover, such a fractional coloring can be computed in polynomial time.*

A slightly better upper bound of $1.367L$ has been proved in [13] for the cost of the fractional coloring of symmetric sets of paths of load L on binary bidirected trees.

5.2 Applications to Path Coloring on Bounded-Degree Bidirected Trees

In this section we briefly outline how to exploit the (optimal) solution for fractional path coloring which can be obtained in polynomial time for bounded-degree bidirected trees to design a randomized algorithm with approximation ratio better than 5/3.

Given a solution $\bar{x}$ of the fractional path coloring of the set of directed paths P on a bidirected tree T, the idea is to perform a randomized rounding to $\bar{x}$ and obtain an integral solution x. After rounding, x is not a feasible solution to the path coloring problem since some of the constraints of the form $\sum_{I \in \mathcal{I}: p \in I} x(I) \geq 1$ may be violated. However, this is a feasible solution to the path coloring problem on the set of paths $P' \subseteq P$, defined as the set of paths contained in independent sets I such that $x(I) = 1$. This means that we have properly colored some of the paths of P with $w_f(P)$ colors.

Following the analysis in [8] (similar arguments are used in [27] for the analysis of a path coloring algorithm in rings), we can show that if $L = \Omega(\log n)$, where n is the number of nodes in T, then after the rounding procedure the load of paths in $P \backslash P'$, i.e., the load of the paths not colored, is at most $\frac{L}{e} + o(L)$, with high probability. Now, we can apply the algorithm of [15] to color the paths in $P \backslash P'$ with $\frac{5L}{3e} + o(L)$ additional colors. In total, we use at most

$$w_f(P) + \frac{5L}{3e} + o(L)$$

colors. Since both the load L and the fractional path chromatic number of P are lower bounds on the optimal number of colors sufficient for coloring P, we obtain the following result.

Theorem 19 (Caragiannis *et al.* [8]). *There exists a randomized path coloring algorithm, which, given a set of directed paths of load $L = \Omega(\log n)$ on a bidirected tree of bounded degree with n nodes, computes an $1.613+o(1)$-approximate path coloring for P, with high probability.*

In a recent work [9], a generalization of the randomized rounding technique used in [8,27] is proposed. It uses a parameter $q \in (0, 1]$, and, given a set of directed paths P of load $L = \Omega(\log n)$ on a bidirected tree with n nodes and a fractional coloring of P of cost $w_f(P)$, it computes a coloring of some of the paths in P with $q \cdot w_f(P)+o(L)$ colors so that the load of the subset of P consisting of the paths being uncolored is at most $e^{-q} \cdot w_f(P) + o(L)$, with high probability. For sets of paths on bidirected trees of bounded degree, applying this randomized rounding technique with parameter $q = \ln \frac{5}{3}$ and using the algorithm of [15] to color the uncolored paths we obtain a $1.511 + o(1)$-approximation algorithm. The restriction on the load is not very important since sets of paths of load $O\left(\frac{\log n}{\log \log n}\right)$ can be optimally colored in bidirected trees of bounded degree. Furthermore, the technique can be applied with $q = \ln \frac{7}{5}$ to sets of paths on binary bidirected trees with similar restrictions to those in Theorem 16 and, combined with the algorithm of [2], it yields a $1.336 + o(1)$-approximation path coloring algorithm on binary bidirected trees.

6 On-Line Algorithms

In the *on-line* version of the path coloring problem in trees, the input instance is a sequence of paths $P = \{p_1, ..., p_{|P|}\}$ arriving over time. An on-line path coloring algorithm must color the paths following the order of the sequence. For each path, the algorithm has to assign it a color so that no two paths including the same edge of the tree are assigned the same color. Once a path has been colored, it cannot be recolored.

The performance of an on-line algorithm is measured by comparing it to the performance (in our case, the number of colors) of the *off-line* algorithm, i.e., the algorithm that knows the sequence of paths in advance. We say that an on-line algorithm $\mathcal{A}_{on}$ for the path coloring problem is c-competitive (or has competitive ratio c) if, for every instance I, $\mathcal{A}_{on}$ uses at most $c \cdot \chi_{off}$ colors, where χ_{off} is the number of colors used by the optimal off-line algorithm $\mathcal{A}_{off}$. This definition applies to deterministic on-line algorithms. We say that a randomized on-line algorithm $\mathcal{A}_{on}$ is c-competitive if, for every instance I, the expectation of the number of colors used by $\mathcal{A}_{on}$ is at most $c \cdot \chi_{off}$. Usually, we analyze the performance of randomized on-line algorithms against oblivious adversaries, i.e., we assume that input instances do not depend on the random choices of the algorithm (although they may depend on the probability distribution according to which the random choices of the algorithm are made).

The results presented in the following apply to both undirected and bidirected trees. A natural on-line algorithm is the First-Fit algorithm. Colors are denoted by positive integers. When a new path p arrives, it is assigned the minimum integer not assigned to paths sharing an edge with p.

The performance of the First-Fit algorithm has been studied in the context of on-line coloring of d-inductive graphs [23]. A graph is called d-inductive if its nodes can be numbered in such a way that each node has at most d adjacent nodes with higher numbers. Irani in [23] shows that First-Fit uses at most $O(d \cdot \log n)$ colors for coloring on-line a d-inductive graph with n nodes.

Bartal and Leonardi [4] observed that the conflict graph defined by the paths in an instance of the path coloring problem on a tree is $2(C-1)$-inductive, where C is the clique number of the conflict graph. Using the result of Irani [23], the set of paths can be colored on-line using at most $O(C \cdot \log |P|)$ colors. Note that any algorithm requires at least C colors to color an instance whose conflict graph has a clique of size C even if it knows the paths in advance. Hence, the competitive ratio of the First-Fit algorithm is $O(\log |P|)$. This is logarithmic in the number of nodes of the tree only if $|P|$ is polynomial in n.

Bartal and Leonardi [4] present the following algorithm which is a variation of the First-Fit algorithm. The nodes of the tree are classified into $O(\log n)$ levels so that for any two nodes of the same level i, the path connecting them contains at least one node of level $i-1$. When a path appears, it is first assigned the lowest of the levels of the nodes it touches. The algorithm assigns the path the minimum color not assigned to paths of the same level sharing an edge with p or to any other path of different level. Bartal and Leonardi show that the number of different colors used for on-line path coloring of a set of paths whose conflict graph has clique size C is at most $O(\log n) \cdot C$.

Theorem 20 (Bartal and Leonardi [4]). *There exists an $O(\log n)$-competitive deterministic on-line path coloring algorithm in (undirected or bidirected) trees with n nodes.*

The algorithm presented in [4] is almost optimal within the class of deterministic on-line path coloring algorithms. Indeed, Bartal and Leonardi [4] show the following statement.

Theorem 21 (Bartal and Leonardi [4]). *For any deterministic on-line path coloring algorithm $\mathcal{A}$ in (undirected or bidirected) trees, there exists a (resp. undirected or bidirected) tree T with n nodes and an input instance I on T which can be colored with a constant number of colors, such that $\mathcal{A}$ colors I with at least $\Omega(\log n / \log\log n)$ colors.*

An interesting issue is whether the use of randomization is helpful. Although no randomized algorithm with sublogarithmic competitive ratio is known, Leonardi and Vitaletti [30] have proved the following lower bound for randomized on-line path coloring algorithms.

Theorem 22 (Leonardi and Vitaletti [30]). *For any on-line path coloring algorithm $\mathcal{A}$ in (undirected or bidirected) trees, there exists an oblivious adversary that constructs a (resp. undirected or bidirected) tree T with n nodes and input instances $\mathcal{I}$ on T which can be colored with a constant number of colors, such that the expectation of the number of colors used by $\mathcal{A}$ to color any instance $I \in \mathcal{I}$ is at least $\Omega(\log \log n)$.*

7 Open Problems

Recent work on path coloring in trees has revealed many open problems; some of them are listed below.

– Improving known bounds for the path coloring problem in undirected trees is equivalent to improving known bounds for the edge-coloring problem on multigraphs. Although this appears to be very difficult, it is an intriguing question whether approximation schemes for the problem exist.
– For the path coloring problem in bidirected trees, an open problem is to close the gap between $5L/4$ and $5L/3$ for the number of colors sufficient for coloring sets of directed paths of load L on arbitrary bidirected trees (see Theorems 1 and 12). Closing the gap between $5L/4$ and $7L/5 + o(L)$ for binary trees also deserves some attention.
– Furthermore, although for deterministic greedy path coloring algorithms we know tight bounds on the number of colors, this is not true for randomized greedy algorithms in bidirected trees. Exploring the power of randomized greedy algorithms in more depth is interesting as well.
– Another intriguing open problem is to prove better bounds on the size of the gap between the path coloring and the fractional path coloring in bidirected trees by using different randomized rounding techniques. We believe that this approach is the most promising one for designing better approximation algorithms for the problem.
– Notice that the algorithms proposed are too far from optimality. Although an "absolute" inapproximability result of 4/3 is implied by the NP-completeness results in arbitrary trees, we are not aware of any asymptotic inapproximability results for the problem (i.e., inapproximability results for sets of paths of heavy load). This indicates another direction for future research.
– Also, closing the gap on the competitive ratio between the best known on-line path coloring algorithm in trees and the (randomized) lower bound is another interesting open problem.

References

1. A. Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan. Efficient Routing and Scheduling Algorithms for Optical Networks. *Journal of the ACM*, 43(6), 1996, pp. 973-1001.
2. V. Auletta, I. Caragiannis, C. Kaklamanis, and P. Persiano. Randomized Path Coloring on Binary Trees. *Theoretical Computer Science*, 289(1), pp. 355-399, 2002.

3. Y. Aumann and Y. Rabani. Improved Bounds for All Optical Routing. In *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, 1995, pp. 567-576.
4. Y. Bartal and S. Leonardi. On-Line Routing in All-Optical Networks, *Theoretical Computer Science*, 221(1-2), 1999, pp. 19-39.
5. B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. Graph Problems Arising from Wavelength-Routing in All-Optical Networks. *2nd Workshop on Optics and Computer Science (WOCS '97)*, 1997.
6. C. Berge. Graphs and Hypergraphs. *North-Holland*, 1973.
7. J.-C. Bermond, L. Gargano, S. Perennes, A.A. Rescigno, and U. Vaccaro. Efficient Collective Communication in Optical Networks. *Theoretical Compute Science*, 233 (1-2), 2000, pp. 165-189.
8. I. Caragiannis, A. Ferreira, C. Kaklamanis, S. Perennes, H. Rivano. Fractional Path Coloring with Applications to WDM Networks. In *Proc. of the 28th International Colloquium on Automata, Languages, and Programming (ICALP '01)*, LNCS 2076, Springer, 2001, pp. 732-743.
9. I. Caragiannis and C. Kaklamanis. Approximate Path Coloring with Applications to Wavelength Assignment in WDM Optical Networks. In *Proc. of the 21st International Symposium on Theoretical Aspects of Computer Science (STACS '04)*, LNCS 2996, Springer, pp. 258-269, 2004.
10. I. Caragiannis, C. Kaklamanis, and P. Persiano. Symmetric Communication in All-Optical Tree Networks. *Parallel Processing Letters*, 10(4), 2000, pp. 305-314.
11. I. Caragiannis, C. Kaklamanis, and P. Persiano. Edge Coloring of Bipartite Graphs with Constraints. *Theoretical Computer Science*, 270(1-2), pp. 361-399, 2002.
12. I. Caragiannis, C. Kaklamanis, and P. Persiano. Bounds on Optical Bandwidth Allocation in Directed Fiber Tree Topologies. *2nd Workshop on Optics and Computer Science*, 1997.
13. I. Caragiannis, C. Kaklamanis, P. Persiano, and A. Sidiropoulos. Fractional and Integral Coloring of Locally-Symmetric Sets of Paths on Binary Trees. In *Proc. of the 1st International Workshop on Approximation and Online Algorithms (WAOA '03)*, LNCS 2909, Springer, pp. 81-94, 2003.
14. T. Erlebach and K. Jansen. The Complexity of Path Coloring and Call Scheduling. *Theoretical Computer Science*, 255, pp. 33-50, 2001.
15. T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal Wavelength Routing on Directed Fiber Trees. *Theoretical Computer Science* 221(1-2), 1999, pp. 119-137.
16. M.R. Garey, D.S. Johnson, G.L. Miller, and C.H. Papadimitriou. The Complexity of Coloring Circular Arcs and Chords. *SIAM Journal of Algebraic Discrete Methods*, 1(2), pp. 216-227, 1980.
17. L. Gargano, P. Hell, and S. Perennes. Colouring All Directed Paths in a Symmetric Tree with Applications to WDM Routing. In *Proc. of the 24th International Colloquium on Automata, Languages, and Programming (ICALP '97)*, LNCS 1256, Springer, pp. 505-515, 1997.
18. M.C. Golumbic and R.E. Jamison. The Edge Intersection Graphs of Paths in a Tree. *Journal of Combinatorial Theory Series B*, 38(1), pp. 8-22, 1985.
19. P. E. Green. Fiber-Optic Communication Networks. *Prentice-Hall*, 1992.
20. M. Grötschel, L.Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1, pp. 169-197, 1981.
21. I. Holyer. The NP-Completeness of Edge Coloring. *SIAM Journal of Computing*, 10(4), pp. 718-720, 1981.

22. K. Jansen. Approximation Results for Wavelength Routing in Directed Binary Trees. *2nd Workshop on Optics and Computer Science*, 1997.
23. S. Irani. Coloring Inductive Graphs On-line, *Algorithmica*, 11(1), pp. 53-72, 1994.
24. I. A. Karapetian. On the Coloring of Circular Arc Graphs. *Akad. Nauk Armeyan SSR Doklady*, 70(5), pp. 306-311, 1980 (in Russian).
25. R. Klasing. Methods and Problems of Wavelength-Routing in All-Optical Networks. In *Proc. of MFCS 98 Workshop on Communication*, pp. 1-9, 1998.
26. J. Kleinberg and E. Tardos. Disjoint Paths in Densely Embedded Graphs. In *Proc. of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS '95)*, pp. 52-61, 1995.
27. V. Kumar. An Approximation Algorithm for Circular Arc Coloring. *Algorithmica*, 30(3), pp. 406-417, 2001.
28. S.R. Kumar, R. Panigrahy, A. Russell, and R. Sundaram. A Note on Optical Routing in Trees, *Information Processing Letters*, 62, pp. 295-300, 1997.
29. E. Kumar and E. Schwabe. Improved Access to Optical Bandwidth in Trees. In *Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, pp. 437-444, 1997.
30. S. Leonardi and A. Vitaletti. Randomized Lower Bounds for On-line Path Coloring. In *Proc. of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98)*, LNCS 1518, Springer, pp. 232-247, 1998.
31. D. Minoli. Telecommunications Technology Handbook, *Artech House*, 1991.
32. T. Nishizeki and K. Kashiwagi. On the 1.1 Edge-Coloring of Multigraphs. *SIAM Journal of Discrete Mathematics*, 3(3), pp. 391-410, 1990.
33. R. Pankaj. Architectures for Linear Lightwave Networks. PhD. Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge MA, 1992.
34. Y. Rabani. Path Coloring on the Mesh. In *Proc. of the 37th IEEE Symposium on Foundations of Computer Science (FOCS '96)*, pp. 400-409, 1996.
35. P. Raghavan and E. Upfal. Efficient Routing in All-Optical Networks. In *Proc. of the 26th Annual ACM Symposium on the Theory of Computing (STOC '94)*, pp. 133-143, 1994.
36. R. Ramaswami and K. Sivarajan. Optical Networks: A Practical Perspective. *Morgan Kauffman Publishers*, 1998.
37. C.E. Shannon. A Theorem on Colouring Lines of a Network. *J. Math. Phys.*, 28, pp. 148-151, 1949.
38. A. Tucker. Coloring a Family of Circular Arcs. *SIAM Journal on Applied Mathematics*, 29(3), 493-502, 1975.

Approximation Algorithms for Edge-Disjoint Paths and Unsplittable Flow

Thomas Erlebach[⋆]

Department of Computer Science, University of Leicester, UK
t.erlebach@mcs.le.ac.uk

Abstract. In the maximum edge-disjoint paths problem (MEDP) the input consists of a graph and a set of requests (pairs of vertices), and the goal is to connect as many requests as possible along edge-disjoint paths. We give a survey of known results about the complexity and approximability of MEDP and sketch some of the main ideas that have been used to obtain approximation algorithms for the problem. We consider also the generalization of MEDP where the edges of the graph have capacities and each request has a profit and a demand, called the unsplittable flow problem.

1 Introduction

Optimization problems concerning edge-disjoint paths in graphs have led to a number of interesting approximation results in the last decade. One of the main reasons for studying such problems is that they are encountered in modern communication networks where establishing a connection requires reserving a certain amount of bandwidth on all edges along some path from the sender to the receiver. If the network does not have sufficient bandwidth to satisfy all connection requests, some requests must be rejected and it is meaningful to try to maximize the number of accepted requests. The optimization problem that seems to lie at the heart of this call admission control problem is the maximum edge-disjoint paths problem (MEDP).

An instance of MEDP is given by a directed graph $G = (V, E)$ and a set (or multiset) containing k requests $\mathcal{R} = \{(s_i, t_i) \mid i = 1, \ldots, k\}$, where each request is a pair of vertices in V. The request (s_i, t_i) asks for a directed path from s_i to t_i in G. We often use "request i" and "request (s_i, t_i)" interchangeably. A feasible solution is given by a subset $\mathcal{A}$ of $\mathcal{R}$ and an assignment of edge-disjoint paths to all requests in that subset. More precisely, each $(s_i, t_i) \in \mathcal{A}$ must be assigned a directed path π_i from s_i to t_i in G such that no two paths π_i and π_j (for $i, j \in \mathcal{A}$ and $i \neq j$) have a directed edge of the graph in common. Such a subset $\mathcal{A}$ is called *realizable* and a set of edge-disjoint paths assigned to the

⋆ Work partly done while the author was affiliated with ETH Zurich and supported by EU Thematic Network APPOL II (IST-2001-32007), with funding provided by the Swiss Federal Office for Education and Science, and by the Swiss National Science Foundation under Contract No. 2100-63563.00 (AAPCN).

requests in $\mathcal{A}$ is called an *edge-disjoint routing* for $\mathcal{A}$. The goal is to maximize the cardinality of $\mathcal{A}$, denoted by $|\mathcal{A}|$. The requests in $\mathcal{A}$ are called the *accepted requests*, those in $\mathcal{R} \setminus \mathcal{A}$ the *rejected requests*.

MEDP can also be studied for undirected graphs. In this case, a request (s_i, t_i) asks for an undirected path connecting s_i and t_i in the given undirected graph, and two paths are edge-disjoint if they do not share an undirected edge. In any case, MEDP can be stated in a compact way as follows.

Problem MEDP (MAXIMUM EDGE-DISJOINT PATHS)
Input: graph $G = (V, E)$, requests $\mathcal{R} = \{(s_i, t_i) \mid i = 1, \dots, k\}$
Feasible solution: realizable subset $\mathcal{A} \subseteq \mathcal{R}$ and edge-disjoint routing for $\mathcal{A}$
Goal: maximize $|\mathcal{A}|$

In general, the same vertex may appear several times as an endpoint of a request in $\mathcal{R}$, and we may even have several identical requests in $\mathcal{R}$. On the other hand, if a set $\mathcal{R}$ of requests has the property that every vertex of G is the endpoint of at most one request in $\mathcal{R}$, the set $\mathcal{R}$ is called a *partial permutation*.

As it turns out that MEDP is an $\mathcal{NP}$-hard optimization problem in general, one is interested in identifying special cases that can be solved optimally in polynomial time and in deriving good approximation algorithms for the other cases. An algorithm for MEDP is a ρ-approximation algorithm if it runs in polynomial time and always outputs a feasible solution $\mathcal{A}$ satisfying $|\mathcal{A}| \geq OPT/\rho$, where OPT denotes the cardinality of an optimal solution. For randomized algorithms, the value $|\mathcal{A}|$ in this definition is replaced by its expectation, taken over the random choices of the algorithm. The approximation ratio ρ can also be a function of certain parameters of the given instance, for example, of the number of vertices or edges of the given graph. We always let $n = |V|$ and $m = |E|$.

Some of the algorithms that we will encounter work also for the *on-line* version of the problem, where the requests are presented to the algorithm one by one in arbitrary order and the algorithm must accept or reject each request without knowledge of future requests. In this case, we will say that the algorithm is an *on-line algorithm*.

In this chapter, we give a tutorial survey of known results concerning approximation algorithms for MEDP. We treat results for arbitrary graphs as well as results for specific graph classes. In Section 1.1, we define all the specific graph classes that we deal with as well as some graph parameters. In Section 1.2, we give a brief survey of known complexity results for MEDP, i.e., we explain which special cases can be solved optimally in polynomial time and which variants are known to be $\mathcal{NP}$-hard or $\mathcal{APX}$-hard.

Section 2 discusses approximation algorithms for MEDP in arbitrary graphs. Sections 2.1 to 2.3 analyze various greedy-type algorithms, some of which achieve approximation ratio $O(\sqrt{m})$. In Section 2.4, we give an inapproximability result showing that no better approximation ratio (as a function of m) can be achieved for arbitrary directed graphs. Section 2.5 deals with the linear programming relaxation of MEDP and shows that although the integrality gap can be very large in general, randomized rounding yields a constant-factor approximation algorithm for a generalization of MEDP with large edge capacities.

Results for specific graph classes are reviewed in Section 3. We discuss results for trees, trees of rings, meshes, hypercubes, de Bruijn graphs, expanders, and complete graphs. In Section 4 we briefly comment on the version of MEDP with pre-determined paths. In Section 5 we consider a generalization of MEDP with edge capacities, demand values, and profit values, called the *unsplittable flow problem*. We present combinatorial algorithms for arbitrary directed graphs and give a summary of results that have been achieved using linear programming techniques. In Section 6, we mention further results that are related to MEDP and the unsplittable flow problem.

1.1 Definition of Graph Classes and Graph Parameters

We give brief definitions of the graph classes that we consider later on. First, consider undirected graphs. A graph is a *complete graph* or a *clique* if there is an edge between every pair of vertices. A *chain* is a graph that consists of a single path. A *tree* is a connected acyclic graph. A tree in which all vertices except one have degree 1 is called a *star*. A *spider* or *generalized star* is a tree in which at most one vertex can have degree larger than 2. A *ring* is a graph that consists of a single cycle. A connected graph all of whose biconnected components are rings is called a *tree of rings*.

The (two-dimensional) $n_1 \times n_2$ *mesh* is the graph with vertex set $V = \{(x, y) \in \mathbb{N}^2 \mid 1 \leq x \leq n_1, 1 \leq y \leq n_2\}$ and with vertex (x, y) being adjacent to the vertices in $\{(x-1, y), (x+1, y), (x, y+1), (x, y-1)\} \cap V$. While all internal vertices have degree 4, the vertices on the boundary of the mesh have only two or three neighbors.

The *hypercube* of dimension d is the graph with 2^d vertices corresponding to the 2^d binary strings of length d, and with an edge between two vertices if and only if the respective binary strings differ only in one position. There are several constant-degree graphs that are called *hypercubic networks*: The d-dimensional *butterfly* is a graph whose vertices are pairs (w, i), where i is an integer satisfying $0 \leq i \leq d$ and w is a binary string of length d. For a vertex (w, i), we call i its *layer*. There is an edge between two vertices (w, i) and (u, j) if and only if $j = i+1$ and either $w = u$ or w and u differ precisely in the jth bit from the left. The binary d-dimensional *de Bruijn* graph is a directed graph with 2^d vertices corresponding to the 2^d binary strings of length d. It has a directed edge from u to v if and only if the binary representation of v can be obtained from the binary representation of u by a cyclic left-shift or by a cyclic left-shift followed by flipping the rightmost bit. An undirected de Bruijn graph is obtained from a directed de Bruijn graph by ignoring the directions of the edges and removing self-loops.

A *bidirected* graph is a graph that is obtained from an undirected graph by replacing each undirected edge with two directed edges of opposite directions. For any of the graph classes defined above, we can thus also consider its bidirected counterpart.

The maximum degree of a graph G is denoted by $\Delta(G)$. For a given graph $G = (V, E)$ and a set $X \subseteq V$, we denote by $\delta(X)$ the set of edges with one endpoint in X and the other endpoint in $V \setminus X$. The *expansion* (or *flux*) $\beta(G)$ of the graph G is defined as

$$\beta(G) = \min_{X \subset V, |X| \le \frac{|V|}{2}} \frac{|\delta(X)|}{|X|}.$$

A graph is called an *expander* if its expansion is bounded from below by some constant.

If the graph G is clear from the context, we write Δ and β for $\Delta(G)$ and $\beta(G)$, respectively.

The *treewidth* of a graph with n nodes is a value between 1 and $n-1$ that measures (in some sense) how "tree-like" a graph is. For example, a graph has treewidth 1 if and only if all its connected components are trees. We do not give the definition of treewidth here and refer the reader to [7,8].

1.2 Polynomial-Time Solvable Cases and Hardness Results

Most of the early work on edge-disjoint paths problems has focused on the version of the problem where the goal is to either route *all* given requests along edge-disjoint paths or certify that such a routing does not exist. This decision problem is one of the classical $\mathcal{NP}$-complete problems [27]. For directed graphs, Fortune, Hopcroft, and Wyllie [23] proved that the problem is $\mathcal{NP}$-complete even if only two terminal pairs are given. The results in the graph minor series by Robertson and Seymour led to a polynomial algorithm for undirected graphs and any constant number of terminal pairs [51]. If the number of terminal pairs is arbitrary, the problem was shown $\mathcal{NP}$-complete for meshes in [42] and in planar graphs of maximum degree 3 by Middendorf and Pfeiffer [46]. Substantial effort has been devoted to the identification of polynomial-time solvable special cases; we refer the reader to the surveys by Frank [24] and Vygen [57].

An intensive investigation of the maximization version MEDP started in the 1990s and is still continuing. Observe that there are classes of graphs for which it can be checked in polynomial time whether a set $\mathcal{R}$ of requests is realizable, but if the answer is *no*, it is $\mathcal{NP}$-hard to compute a maximum subset of realizable requests. Bidirected trees and undirected or bidirected trees of rings are examples.

Nevertheless, MEDP can be solved optimally in polynomial time for some classes of graphs. A first such class are chains. The routing for each request in a chain is uniquely determined by its endpoints (the same is true for arbitrary trees). Therefore, a set of requests in a chain can be represented as a set of intervals on the real line, and it suffices to find a maximum number of pairwise disjoint intervals. This problem has been studied in the context of interval graphs and is known to be solvable in linear time [30]. Thus, MEDP can be solved optimally in polynomial time for undirected or bidirected chains.

For undirected trees, a polynomial algorithm for MEDP has been found by Garg, Vazirani and Yannakakis [28].

For bidirected trees of arbitrary degree, MEDP has been proved $\mathcal{APX}$-complete by Erlebach and Jansen [21], implying that there is a constant $r > 1$ such that no polynomial-time algorithm can achieve approximation ratio r unless

$\mathcal{P} = \mathcal{NP}$. The proof is based on the proof by Garg, Vazirani, and Yannakakis [28] showing that a generalization of MEDP with edge capacities is $\mathcal{APX}$-complete in undirected trees with edge capacities in $\{1, 2\}$. It can also be adapted to trees of rings, showing that MEDP is $\mathcal{APX}$-complete for undirected and bidirected trees of rings [20].

In bidirected stars, each request uses at most two edges, and the MEDP problem can be reduced to the maximum bipartite matching problem. This approach can be extended to give a polynomial-time algorithm for bidirected spiders as well, as shown for a more general problem by Erlebach and Vukadinović [22]. In undirected or bidirected rings, MEDP can also be solved optimally in polynomial time [58,49].

It is known that many hard problems can be solved efficiently on graphs of bounded treewidth [7]. However, since trees of rings have treewidth 2, MEDP is $\mathcal{APX}$-hard even for graphs with treewidth 2. On the other hand, it is remarked in [19] that MEDP can be solved optimally in polynomial time on graphs whose treewidth and maximum degree are both bounded by a constant. Furthermore, Zhou et al. [59] have shown that MEDP can be solved optimally in polynomial time on graphs of bounded treewidth if the number of requests is $O(\log n)$ or if certain conditions on the location of the request endpoints are fulfilled (including the case when the union of the graph and the demand graph has bounded treewidth, where the demand graph is the graph with an edge between the endpoints of each request).

It is interesting to note that for undirected graphs, no stronger inapproximability result than $\mathcal{APX}$-hardness is known for MEDP. For directed graphs, we will see a strong inapproximability result in Section 2.4.

2 Edge-Disjoint Paths in Arbitrary Graphs

2.1 The Greedy Algorithm

A very natural algorithm for MEDP is the greedy algorithm (see Fig. 1). It processes the requests in arbitrary order. For each request, it checks whether the request can be routed along a path that is edge-disjoint from all paths assigned

```
algorithm Greedy($G = (V, E), \mathcal{R} = \{(s_i, t_i) \mid i = 1, \ldots, k\}$):
  $\mathcal{A} \leftarrow \emptyset$;
  for $i = 1$ to $k$ do
      if $\exists$ path from $s_i$ to $t_i$ in $G$ then
          $\mathcal{A} \leftarrow \mathcal{A} \cup \{(s_i, t_i)\}$;
          $\pi_i \leftarrow$ a shortest path from $s_i$ to $t_i$ in $G$;
          remove all edges of $\pi_i$ from $G$;
      fi;
  od;
  return $\mathcal{A}$ and $\{\pi_i \mid (s_i, t_i) \in \mathcal{A}\}$;
```

Fig. 1. The greedy algorithm

to previously accepted requests. If so, it accepts the request and routes it along some shortest path that does not intersect previously accepted paths.

Unfortunately, this algorithm does not achieve a good approximation ratio in general. For example, consider a chain of n vertices, numbered from 1 to n. Assume that the first request processed by the algorithm is $(1, n)$ and that there are $n-1$ further requests of the form $(i, i+1)$, for $i = 1, 2, \ldots, n-1$. The greedy algorithm accepts the first request but no other request, because the first request blocks all $n-1$ edges of the chain. The optimal solution, however, consists of the other $n-1$ requests. Thus, the optimal solution is better than the greedy solution by a factor of $n-1$.

On the other hand, we can show that the optimal solution can never be better than the greedy solution by more than a factor of $n-1$. To see this, we employ the following proof technique. Let $\mathcal{A}^*$ be the set of requests in an arbitrary optimal solution and let π_i^* be the path assigned to request $(s_i, t_i) \in \mathcal{A}^*$ by this solution. Let $\mathcal{A}$ and π_i be defined analogously for the greedy solution. We consider the execution of the greedy algorithm and, whenever it accepts a request i, we remove from $\mathcal{A}^*$ the request i (provided $i \in \mathcal{A}^*$) and all other requests in $\mathcal{A}^*$ whose optimal path π_j^* intersects the greedy path π_i. Note that at any time of the execution, the paths π_j^* of the requests that remain in $\mathcal{A}^*$ are disjoint from the paths of the requests that have already been accepted by the greedy algorithm. When the greedy algorithm terminates, the set $\mathcal{A}^*$ must be empty, by definition of the greedy algorithm. Therefore, if we can show that for each request accepted by the greedy algorithm, we have to remove at most ρ paths from the optimal solution, this implies that $|\mathcal{A}^*| \leq \rho \cdot |\mathcal{A}|$ and, consequently, the greedy algorithm achieves approximation ratio at most ρ.

So how many requests do we have to remove from $\mathcal{A}^*$ when the greedy algorithm accepts a request i? In a graph with n vertices, every simple path (and therefore also π_i) consists of at most $n-1$ edges. Consequently, it can intersect at most $n-1$ paths π_j^* of requests in $\mathcal{A}^*$. Since it suffices to remove these up to $n-1$ paths as well as the request i itself, this shows that we have to remove at most n paths from $\mathcal{A}^*$. In fact, we can strengthen this to show that we have to remove at most $n-1$ paths: If the path π_i consists of fewer than $n-1$ edges, the above argument can be used. If π_i has length $n-1$, it does in fact pass through all vertices of the graph. We would have to remove n paths from $\mathcal{A}^*$ only if π_i^* was edge-disjoint from π_i, but this cannot happen since π_i passes through all vertices and is a shortest path from s_i to t_i (among all paths that do not intersect previously accepted paths). Thus, the approximation ratio of the greedy algorithm is bounded from above by $n-1$ as well.

Theorem 1. *The greedy algorithm has approximation ratio $n-1$ for* MEDP *in directed or undirected graphs with n vertices, and this bound is tight.*

2.2 The Bounded-Length Greedy Algorithm

The greedy algorithm can perform badly because it may accept a request and route it on a very long path while the optimal solution accepts many requests

```
algorithm $BoundedGreedy(G = (V,E), \mathcal{R} = \{(s_i,t_i) \mid i = 1,\ldots,k\}, D)$:
  $\mathcal{A} \leftarrow \emptyset$;
  for $i = 1$ to $k$ do
      if $\exists$ path of length at most $D$ from $s_i$ to $t_i$ in $G$ then        (*)
          $\mathcal{A} \leftarrow \mathcal{A} \cup \{(s_i,t_i)\}$;
          $\pi_i \leftarrow$ a shortest path from $s_i$ to $t_i$ in $G$;
          remove all edges of $\pi_i$ from $G$;
      fi;
  od;
  return $\mathcal{A}$ and $\{\pi_i \mid (s_i,t_i) \in \mathcal{A}\}$;
```

Fig. 2. The bounded-length greedy algorithm. It differs from the standard greedy algorithm only by the modified condition in line (*).

whose paths intersect the long path instead. One idea to avoid this problem is to restrict the length of the paths that the greedy algorithm is allowed to use. The bounded-length greedy algorithm, suggested by Kleinberg [33] and shown in Fig. 2, takes an additional parameter D and behaves like the greedy algorithm, but accepts a request only if it can be routed on a path of length at most D. Thus, each accepted request can intersect the paths of at most D other requests in the optimal solution. On the other hand, the algorithm rejects all requests whose endpoints are at distance larger than D. This means that the algorithm will have approximation ratio ∞ on instances where all requests have endpoints at distance larger than D. Therefore, we need to check for this case separately and arrive at the following algorithm, which we call *BoundedGreedy*$'$.

1. If for all requests (s_i, t_i) the length of a shortest path from s_i to t_i is at least $D + 1$, return the solution consisting of a single request (s_i, t_i) and an arbitrary path from s_i to t_i assigned to it.
2. Otherwise, run $BoundedGreedy(G, \mathcal{R}, D)$.

To analyze this algorithm, let us first consider the case that the condition of step 1 applies and that the algorithm accepts a single request. Since all paths assigned to a request must use at least $D+1$ of the m edges of the graph, even the optimal solution can contain at most $m/(D+1)$ paths. Thus, the ratio is at most $m/(D+1)$ in this case. Now assume that the algorithm *BoundedGreedy* is actually called. Each request accepted by *BoundedGreedy* (and thus also by *BoundedGreedy*$'$) blocks at most D other requests that could have been accepted instead. All requests in the optimal solution that are routed along a path of length at most D in the optimal solution must either be accepted by the algorithm or intersect a path accepted by the algorithm. Thus, we can apply the proof technique of the previous section to show that the optimal solution contains at most $(D + 1)|\mathcal{A}|$ requests with paths of length at most D, where $\mathcal{A}$ is the set of requests accepted by *BoundedGreedy*$'$. On the other hand, by the same argument as above, the number of paths in the optimal solution that are routed along paths of length at least $D + 1$ is at most $m/(D+1)$. Thus, the optimal solution contains at most $m/(D+1)+(D+1)|\mathcal{A}| \leq$

$(m/(D+1)+D+1)|\mathcal{A}|$ paths. This proves that *BoundedGreedy'* achieves approximation ratio at most $D+1+m/(D+1)$. We can choose $D = \lceil\sqrt{m}\,\rceil - 1$ to obtain a $2\lceil\sqrt{m}\,\rceil$-approximation algorithm for MEDP.

To construct a bad instance for *BoundedGreedy'*, take a ring with m edges and vertices, the vertices being numbered clockwise from 1 to m. Choose m such that $\sqrt{m}$ is integral. Consider the set of requests containing the *first* request $(1, \sqrt{m})$, the *short* requests $(j, j+1)$ for $j = 1, \ldots, \sqrt{m}-1$, and the *long* requests $(i\sqrt{m}, (i+1)\sqrt{m})$ for $i = 1, \ldots, \sqrt{m}-1$. The algorithm accepts the first request $(1, \sqrt{m})$ and routes it along the clockwise path. All other requests cannot be routed along a path of length at most $D = \sqrt{m}-1$ now, so the algorithm rejects all other requests. On the other hand, the optimal solution accepts all short requests and all long requests, thus giving a solution with $2\sqrt{m} - 2$ requests.

Theorem 2 (Kleinberg, 1996). *Algorithm BoundedGreedy' with parameter* $D = \lceil\sqrt{m}\,\rceil - 1$ *has approximation ratio* $O(\sqrt{m})$ *for* MEDP *in directed or undirected graphs with* m *edges.*

We will see later that *BoundedGreedy* achieves a much better approximation ratio in expanders and other specific classes of graphs. Furthermore, it is interesting to note that *BoundedGreedy'* can be adapted to the on-line setting by flipping a coin in the beginning and running *BoundedGreedy* with probability $\frac{1}{2}$ and the standard greedy algorithm otherwise.

Routing Number and Flow Number. Kolman and Scheideler [40] analyze the approximation ratio achievable by the bounded-length greedy algorithm for undirected graphs in terms of the *routing number* of the given graph. Consider an undirected graph $G = (V, E)$ with n nodes. For a set P of paths in G, the congestion of an edge $e \in E$ is the number of paths containing e. The congestion of P is the maximum congestion among all edges. Denote by S_n the set of all permutations from V to V. For any permutation $\pi \in S_n$ and any L that is at least the diameter of G, let $C(G, L, \pi)$ be the minimum congestion among all path sets that realize π (i.e., contain a path from v to $\pi(v)$ for all $v \in V$) and that consist of paths with length at most L. Then the *L-bounded routing number* $R(G, L)$ of G is defined by

$$R(G, L) = \max_{\pi \in S_n} \max\{C(G, L, \pi), L\}\,.$$

Intuitively, if a graph has L-bounded routing number R, then for any permutation of the vertices, there exists a set of paths with length at most L and congestion at most R that realizes the permutation.

The *(unbounded) routing number* $R(G)$ is defined as $R(G) = \min_L R(G, L)$. For any graph G, $R(G)$ lies between $\Theta(\beta(G)^{-1})$ and $O(\Delta\beta(G)^{-1}\log n)$, where Δ is the maximum degree of G and $\beta(G)$ is the expansion as defined in Section 1.1. There is a constant-factor approximation algorithm for computing the routing-number of a given graph [52]. The routing number is $\Theta(n)$ for the chain, $\Theta(\sqrt{n})$ for the $\sqrt{n} \times \sqrt{n}$-mesh, and $\Theta(\log n)$ for the butterfly, the hypercube, and constant-degree expanders.

```
algorithm ShortestFirstGreedy($G = (V, E), \mathcal{R} = \{(s_i, t_i) \mid i = 1, \ldots, k\}$):
  $\mathcal{A} \leftarrow \emptyset$;
  while $\mathcal{R}$ contains a request that can be routed in $G$ do
      $(s_i, t_i) \leftarrow$ a request in $\mathcal{R}$ such that the shortest path from $s_i$ to $t_i$ in $G$
                  has minimum length among all requests in $\mathcal{R}$;
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(s_i, t_i)\}$;
      $\mathcal{R} \leftarrow \mathcal{R} \setminus \{(s_i, t_i)\}$;
      $\pi_i \leftarrow$ a shortest path from $s_i$ to $t_i$ in $G$;
      remove all edges of $\pi_i$ from $G$;
  od;
  return $\mathcal{A}$ and $\{\pi_i \mid (s_i, t_i) \in \mathcal{A}\}$;
```

Fig. 3. The shortest-path-first greedy algorithm

For undirected graphs with maximum degree Δ and routing number R, Kolman and Scheideler show that the bounded-length greedy algorithm with length parameter $D = 2R$ achieves approximation ratio at most $(\Delta + 4)R + 1 = O(\Delta R) = O(\Delta^2 \beta(G)^{-1} \log n)$. For graphs with bounded degree, this gives approximation ratio $O(R) = O(\beta(G)^{-1} \log n)$. If the exact value of R is not known, the bounded-length greedy algorithm can be run for all values $D = 2^p$ for $p = 0, 1, \ldots, \log n$, and the best solution can be taken as the output of the algorithm. Thus, approximation ratio $O(\Delta R)$ is achieved also in this case.

In addition, Kolman and Scheideler present a randomized approximation algorithm with ratio $O(\Delta\sqrt{LR})$ for MEDP in undirected graphs with maximum degree Δ and L-bounded routing number R [40]. The algorithm works in the online setting. Note that the bound $O(\Delta\sqrt{LR})$ can be substantially better than the bound $O(\Delta R(G))$. The idea behind the algorithm is that in many graphs, some edges are "bottleneck" edges and some are not. A single path passing through many bottleneck edges could cause the rejection of many subsequent requests. The algorithm thus puts a stricter bound on the number of bottleneck edges of a path than on the number of non-bottleneck edges. The tricky part of the algorithm is the initial phase in which the bottleneck edges are determined.

In subsequent work, Kolman and Scheideler [41] consider the *flow number* $F(G)$ of the given graph G instead of the routing number. They show that the bounded-length greedy algorithm with length parameter $D = 4F(G)$ achieves approximation ratio $O(F(G)) = O(\Delta\beta(G)^{-1} \log n)$ for MEDP in undirected graphs with maximum degree Δ and expansion $\beta(G)$.

It is interesting to note that $F(G)$ is 1 for complete graphs, $\Theta(\log n)$ for hypercubes and expanders, $\Theta(\sqrt{n})$ for the $\sqrt{n} \times \sqrt{n}$-mesh, and $\Theta(n)$ for the chain. As the flow number is defined in the more general context of graphs with edge capacities, we give its definition only in Section 5.1 where we use it in the context of the unsplittable flow problem.

The main idea underlying the use of the flow number is that the bounded-length greedy algorithm has a good approximation ratio if there exists an optimal solution that consists of short paths only. The key ingredient of the analysis is then a *shortening lemma*, which implies that any set of disjoint paths in a graph with flow number F can be converted into a set of fractional flows such that the load on every edge is at most 2 and all flow paths have length at most $4F$. The shortening lemma is applied to the paths in the optimal solution of the given instance of MEDP, and then the size of the solution produced by the bounded-length greedy algorithm is related to the fractional flows with short flow paths.

Kolman and Scheideler [41] use this analysis to obtain approximation ratio $16F + 1$ for the unsplittable flow problem (that includes MEDP as a special case), and it is not difficult to verify that the ratio for MEDP can even be bounded by $8F + 1$ [13].

2.3 The Shortest-Path-First Greedy Algorithm

Another modification of the greedy algorithm is the shortest-path-first greedy algorithm, shown in Fig. 3. It was suggested by Kolliopoulos and Stein [37]. Instead of processing the given requests sequentially in an arbitrary order, the algorithm *ShortestFirstGreedy* considers all remaining requests and accepts a request whose shortest path has length ℓ only if no other remaining request can be routed along a path of length smaller than ℓ. As a consequence, the algorithm accepts requests in order of non-decreasing lengths of their shortest paths. It is easy to see that the worst-case approximation ratio of *ShortestFirstGreedy* is at least as good as that of *BoundedGreedy*.

Algorithm *ShortestFirstGreedy* achieves approximation ratio $\lceil\sqrt{m}\,\rceil$. To prove this, consider an arbitrary optimal solution $\mathcal{A}^*$ with paths π_i^*. Again, consider an execution of the algorithm and when the algorithm accepts a request i and routes it along a path π_i, remove from $\mathcal{A}^*$ the request i (if $i \in \mathcal{A}^*$) as well as all requests j whose path π_j^* intersects π_i. By the order in which the algorithm accepts the requests, all requests $j \in \mathcal{A}^*$ have length at least $|\pi_i|$ when the algorithm accepts request i. If $|\pi_i| \le \lceil\sqrt{m}\,\rceil - 1$, we have to remove at most $\lceil\sqrt{m}\,\rceil$ paths from $\mathcal{A}^*$, as before. If $|\pi_i| \ge \lceil\sqrt{m}\,\rceil$, there can be at most $\sqrt{m}$ paths left in $\mathcal{A}^*$, because they all have length at least $\lceil\sqrt{m}\,\rceil$. So we have to remove at most $\sqrt{m}$ paths in this case as well. This shows that the approximation ratio of *ShortestFirstGreedy* is at most $\lceil\sqrt{m}\,\rceil$.

Theorem 3 (Kolliopoulos and Stein, 1998). *The shortest-path-first greedy algorithm has approximation ratio at most* $\lceil\sqrt{m}\,\rceil$ *for* MEDP *in directed or undirected graphs with* m *edges.*

We remark that m can be replaced by m^* in the analysis above, where m^* is the number of edges that are actually used by paths in the optimal solution.

By refining the analysis above, we can bound the approximation ratio of *ShortestFirstGreedy* in terms of the average path length $d^* = m^*/|\mathcal{A}^*|$ of an optimal solution $\mathcal{A}^*$ whose paths use m^* edges in total. Let $\mathcal{A}' = \mathcal{A}^* \setminus \mathcal{A}$, i.e., $\mathcal{A}'$ denotes the requests that are in the optimal solution, but not in the solution computed by the algorithm. When the algorithm accepts request i and routes it along path π_i, we remove from $\mathcal{A}'$ the requests whose paths π_j^* intersect π_i. Let k_i denote the number of requests that are removed from $\mathcal{A}'$ because of path π_i. Each of these k_i requests has a path of length at least $|\pi_i|$, and $|\pi_i|$ must be at least k_i. Therefore, these k_i requests use at least k_i^2 of the m^* edges used in the optimal solution, and we get

$$\sum_{i\in\mathcal{A}} k_i^2 \le m^*.$$

Since $\sum_{i\in\mathcal{A}} k_i^2 \ge \left(\sum_{i\in\mathcal{A}} k_i \right)^2 /|\mathcal{A}|$ by the Cauchy-Schwarz inequality, we get $\left(\sum_{i\in\mathcal{A}} k_i \right)^2 /|\mathcal{A}| \le m^*$. Using $\sum_{i\in\mathcal{A}} k_i = |\mathcal{A}'|$, we obtain $|\mathcal{A}| \ge |\mathcal{A}'|^2/m^*$. If $|\mathcal{A}| \ge |\mathcal{A}^*|/2$, the algorithm has ratio at most 2. If $|\mathcal{A}| < |\mathcal{A}^*|/2$, we have $|\mathcal{A}'| = |\mathcal{A}^* \setminus \mathcal{A}| > |\mathcal{A}^*|/2$ and obtain

$$|\mathcal{A}| \ge \frac{|\mathcal{A}'|^2}{m^*} > \frac{|\mathcal{A}^*|^2}{4m^*} = \frac{|\mathcal{A}|^*}{4\frac{m^*}{|\mathcal{A}^*|}} = \frac{|\mathcal{A}|^*}{4d^*}.$$

Thus, the approximation ratio is always bounded by $4d^*$.

Theorem 4 (Kolliopoulos and Stein, 1998). *The shortest-path-first greedy algorithm achieves approximation ratio $O(d^*)$ for* MEDP *on instances for which some optimal solution has average path length d^*.*

We remark that approximation ratio $O(d^*)$ can also be achieved by running *BoundedGreedy* with parameter $D = 1, 2, 4, 8, \ldots, n$ and outputting the best of the solutions obtained in this way. This follows because at least half the paths of the optimal solution have length at most $2d^*$, and *BoundedGreedy* will produce a solution with the required quality for the value of D that lies in the interval $[2d^*, 4d^*)$ (see the full version of [40]).

Chekuri and Khanna [15] showed that the bound of $O(\sqrt{m})$ on the approximation ratio of the shortest-path-first greedy algorithm is not tight for dense graphs. They proved upper bounds of $O(n^{2/3})$ and $O(n^{4/5})$ for undirected and directed graphs, respectively. Their improved result is obtained by tightening the analysis concerning long paths in the optimal solution. In the arguments above, we used the fact that the optimal solution can contain at most m/ℓ paths of length at least ℓ. Roughly speaking, Chekuri and Khanna show that in a graph where the distance between the end vertices of each request is at least ℓ, at most $O(n^2/\ell^2)$ (in the undirected case) and $O(n^4/\ell^4)$ (in the directed case) requests can be connected along edge-disjoint paths. This leads to the better bound in dense graphs. They also show that the approximation ratio of the shortest-path-first greedy algorithm is $\Omega(n^{2/3})$ in directed acyclic graphs and in undirected graphs. The upper bound on the approximation ratio of the shortest-path-first greedy algorithm for directed graphs was subsequently improved to $O((n \log n)^{2/3})$ by Varadarajan and Venkataraman [56]. They obtain this result by showing that in a directed graph where the distance between the end vertices of each request is at least ℓ, at most $O((n^2/\ell^2) \cdot (\log n/\ell)^2)$ of the requests can be connected along edge-disjoint paths.

Theorem 5 (Chekuri and Khanna, 2003; Varadarajan and Venkataraman, 2004). *The approximation ratio of the shortest-path-first greedy algorithm for* MEDP *is* $O(\min\{\sqrt{m}, n^{2/3}\})$ *in undirected graphs and* $O(\min\{\sqrt{m}, (n \log n)^{2/3}\})$ *in directed graphs.*

For directed acyclic graphs, Chekuri and Khanna [15] present a combinatorial algorithm that achieves approximation ratio $O(\sqrt{n} \cdot \log n)$. The algorithm is based on the observation that, if the optimal solution contains many long paths, there must be a vertex that is contained in a significant fraction of these long paths.

2.4 Inapproximability for Directed Graphs

The bounded greedy algorithm and the shortest-path-first greedy algorithm achieve approximation ratio $O(\sqrt{m})$ for directed or undirected graphs with m edges. For directed graphs, an essentially matching inapproximability result has been obtained by Guruswami et al. [31].

Theorem 6 (Guruswami et al., 1999). *For* MEDP *in directed graphs with* m *edges, there cannot be an* $m^{\frac{1}{2}-\varepsilon}$*-approximation algorithm for any* $\varepsilon > 0$ *unless* $\mathcal{P} = \mathcal{NP}$.

Proof. The proof is based on the hardness of the problem 2DirPath, i.e., deciding for a directed graph $H = (V, E)$ and four distinct vertices u_1, w_1, u_2, w_2 whether H contains paths from u_1 to w_1 and from u_2 to w_2 that are edge-disjoint. This problem has been proved $\mathcal{NP}$-complete by Fortune, Hopcroft and Wyllie in [23]. Consider some fixed $\varepsilon > 0$. Given an instance I of the problem 2DirPath, one can efficiently construct an instance of MEDP with k requests in a directed graph G with m edges such that the optimal solution contains all k requests if I is a yes-instance and only one request if I is a no-instance. Therefore, an approximation algorithm for MEDP with ratio smaller than k would allow to decide I in polynomial time, implying that $\mathcal{P} = \mathcal{NP}$.

The construction of G is illustrated in Fig. 4. G is the lower right triangle of a $k \times k$ mesh, with all edges pointing upward or to the right. Every internal vertex v of the mesh is replaced by a copy of H. The head of the incoming horizontal edge of v is attached to u_1, the tail of the outgoing horizontal edge to w_1. Similarly, the vertical edges incident to v are attached to u_2 and w_2. Intuitively, this copy of H allows two paths arriving at v from below and from the left to *cross* (i.e., to leave this copy of H on the top and on the right side, respectively) if and only if I is a yes-instance.

The k requests in G are defined as follows: The vertices at the bottom of the mesh (from left to right) are the sources $s_1, s_2, \ldots, s_k$, and the vertices on the right side of the mesh (from bottom to top) are the destinations $t_1, t_2, \ldots, t_k$. If I is a yes-instance, all k requests can be accepted and routed along the canonical path that goes upward until the destination row is reached and then to the right until the destination is hit. If I is a no-instance, there cannot be edge-disjoint paths for any two requests i and j, $i \neq j$, because such paths would have to cross at some copy of H. Therefore, the optimal solution to the

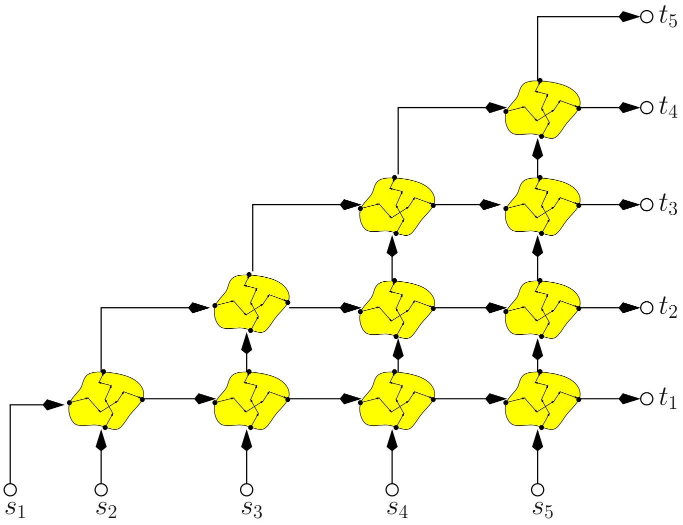

Fig. 4. Construction used in the inapproximability proof

constructed instance of MEDP contains either k requests or only one request, depending on whether I is a yes-instance. If we choose $k = m_H^{1/\varepsilon}$, where m_H is the number of edges of H, the graph G has $m = \Theta(k^2 m_H) = \Theta(k^{2+\varepsilon})$ edges, hence $k = \Theta(m^{1/(2+\varepsilon)})$. Since the construction can be applied for every fixed $\varepsilon > 0$, the theorem follows. □

Note, however, that for the graphs constructed in the proof of Theorem 6, we have $m = O(n)$. Therefore, the lower bound does not necessarily apply to dense graphs. In fact, we have seen in Theorem 5 that the shortest-path-first greedy algorithm indeed achieves approximation ratio strictly better than $O(\sqrt{m})$ in dense graphs. In terms of n, the proof of Theorem 6 gives only $n^{\frac{1}{2}-\varepsilon}$-inapproximability of MEDP in directed graphs.

Another inapproximability result for MEDP in directed graphs has been obtained by Ma and Wang [45]. They prove that MEDP cannot be approximated within ratio $2^{\log^{1-\varepsilon} n}$ in directed graphs with n vertices unless $\mathcal{NP} \subseteq \mathrm{DTIME}(2^{\mathrm{polylog} n})$.

It remains an interesting open problem to determine whether the approximation ratio for MEDP in undirected graphs can be improved substantially. The proof of Theorem 6 cannot be adapted to the undirected case because the variant of 2DirPath for undirected graphs can be solved in polynomial time. In fact, for every constant k there is a polynomial-time algorithm that decides for an undirected graph and k requests whether all k requests can be connected along edge-disjoint paths by the results of Robertson and Seymour [51]. At present, we cannot even exclude the existence of an $O(1)$-approximation algorithm for MEDP in undirected graphs.

2.5 Linear Programming and Randomized Rounding

Linear programming [53] is a very powerful tool in combinatorial optimization. In this section we discuss the natural LP relaxation of MEDP. Intuitively, solutions to the LP relaxation correspond to *fractional* solutions, i.e., solutions that may accept only fractions of certain requests and that may split the accepted fraction of a request among several paths arbitrarily.

In the following, we allow a more general edge capacity constraint than the strict requirement of having edge-disjoint paths: we assume that each edge e has a certain integral capacity $u(e)$ and that a routing is feasible if at most $u(e)$ paths go through every edge e. We call this problem *generalized MEDP*.

An LP Relaxation of MEDP. Let a graph $G = (V, E)$ with edge capacities $u : E \to \mathbb{N}$ and a set of requests $\mathcal{R} = \{(s_i, t_i) \mid i = 1, \ldots, k\}$ be given. Denote by $\mathcal{P}_i$ the set of all paths from s_i to t_i in G. Note that there might be exponentially many paths in $\mathcal{P}_i$; we will discuss later how to deal with this.

It is natural to introduce a variable x_i for each request i, $1 \le i \le k$, that is constrained by $0 \le x_i \le 1$ and represents the fraction of request i that is accepted. Furthermore, a variable $y_{i,\pi}$ is introduced for each path $\pi \in \mathcal{P}_i$; this variable corresponds to the fraction of request i that is routed along path π.

Thus, maximizing the sum of the accepted fractions of the given requests can be formulated as the following linear program:

$$f^* = \max \sum_{i=1}^{k} x_i \tag{1}$$

$$\text{s.t.} \sum_{i,\pi : e \in \pi \in \mathcal{P}_i} y_{i,\pi} \le u(e), \qquad \text{for all } e \in E \tag{2}$$

$$\sum_{\pi \in \mathcal{P}_i} y_{i,\pi} = x_i, \qquad \text{for } 1 \le i \le k \tag{3}$$

$$0 \le x_i \le 1, \qquad \text{for } 1 \le i \le k \tag{4}$$

$$0 \le y_{i,\pi} \le 1, \qquad \text{for } 1 \le i \le k \text{ and } \pi \in \mathcal{P}_i \tag{5}$$

The objective function (1) expresses that the goal is to maximize the sum of the accepted fractions of the requests. Constraint (2) ensures that the edge capacities are not exceeded. Constraint (3) requires that the fractions of request i that are routed along different paths from s_i to t_i add up to x_i, the total accepted fraction of request i. Constraints (4) and (5) ensure that all variables are between zero and one.

The above LP may be of exponential size, but is easy to understand. In order to obtain an LP of polynomial size, one uses variables $f_{i,e}$ for $1 \le i \le k$ and $e \in E$ that represent the fraction of request i that is routed through edge e. The number of such variables is bounded by $k|E|$. Furthermore, the edge capacity constraints (2) are replaced by $\sum_{i=1}^{k} f_{i,e} \le u(e)$ for all $e \in E$, and flow conservation constraints are added for each request i and each vertex $v \in V \setminus \{s_i, t_i\}$ (expressing that the amount of request i reaching v through incoming

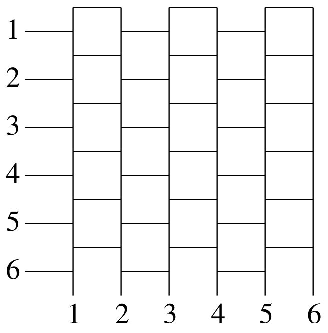

Fig. 5. The brick-wall graph

edges is equal to the amount leaving v through outgoing edges). Constraint (3) is replaced by a constraint ensuring that the amount of request i leaving s_i is equal to x_i. The variables $y_{i,\pi}$ are not needed in this modified LP. The modified LP has polynomial size and can be solved optimally in polynomial time. We refer to the resulting values of the variables $f_{i,e}$ and x_i as $f^*_{i,e}$ and x^*_i, respectively. These quantities can be viewed as representing a *flow* of value x^*_i from s_i to t_i, for $1 \le i \le k$. Using standard flow decomposition methods (also called path stripping) [50], the flow of request i can be efficiently transformed into $O(|E|)$ separate flows along paths from s_i to t_i, and these flows represent a fractional solution to the original (exponential-size) LP with the same objective value. Thus, we can assume from now on that we can compute an optimal fractional solution to the original LP in polynomial time and that this solution has only a polynomial number of variables $y^*_{i,\pi}$ that are non-zero.

A solution to the LP in which the variables x_i and $y_{i,\pi}$ are all integral, i.e., either equal to 0 or equal to 1, corresponds to a feasible solution of the original instance of generalized MEDP. However, adding such integrality constraints makes the linear programming problem $\mathcal{NP}$-hard. Therefore, a meaningful approach is to solve the LP to obtain an optimal fractional solution and then try to convert the fractional solution into an integral solution without losing too much in the objective function. This conversion is usually called *rounding.*

The Integrality Gap. Unfortunately, the gap between the fractional optimum and the integral optimum can be arbitrarily large for the LP formulation of MEDP [28]. A simple example to demonstrate this is shown in Fig. 5. This type of mesh-like graph with internal vertices of degree 3 is often called a *brick-wall graph.* Assume that the graph is undirected, all edges have capacity 1, and there are 6 requests, where both endpoints of request i are labeled by i in the figure. Since any two paths for different requests must cross somewhere and thus share an edge, the integral optimum accepts only one of the six paths. The fractional solution, however, can route 1/2 of each request without violating any capacity constraint, thus giving an objective value of 3. This example can be generalized to give instances where the integral optimum is 1, but the fractional optimum is $\Theta(\sqrt{m})$ for brick-wall graphs with m edges, where m is arbitrarily large. The construction can also be adapted to directed graphs in a straightforward way. Note, however, that the brick-wall graph is sparse and thus the lower bound on the integrality gap in terms of n is only $\Omega(\sqrt{n})$. Chekuri and Khanna [15] prove that an upper bound of $O(\sqrt{n} \cdot \log n)$ on the integrality gap holds for directed, acyclic graphs.

In any case, the discussion above shows that we cannot hope to find a rounding approach that always gives an integral solution with objective value close to the fractional optimum.

Randomized Rounding in the High-Capacity Case. Fortunately, the situation is much better if the edge capacities are required to be large. In this case, the *randomized rounding* approach due to Raghavan and Thompson [50] gives a constant-factor approximation ratio. Our presentation follows the work by Kleinberg [33, pp. 39–41]. The basic idea is to interpret the values of the variables in the fractional solution as probabilities. For the variables x_i, this means that x_i is set to 1 with probability x_i^*. More precisely, we will have to scale down the values x_i^* by some factor μ before the rounding can be done.

Assume that all edge capacities are at least $\varepsilon \log m$ for some constant ε. Let $\mu < 1$ be a positive constant whose choice will be explained later. For each request i, $1 \le i \le k$, make an independent random choice and route request i along path $\pi \in \mathcal{P}_i$ with probability $\mu y^*_{i,\pi}$, for all non-zero $y^*_{i,\pi}$, and reject request i with probability $1 - \sum_{\pi \in \mathcal{P}_i} \mu y^*_{i,\pi} = 1 - \mu x_i^*$.

We need to show that with constant probability, the resulting integral solution does not violate any edge capacity and that its objective value is at least a constant fraction of the fractional optimum. The number of paths going through an edge in the integral solution is the number of fractional paths through that edge that were rounded up. Thus, the random variable representing the number of paths going through the edge in the integral solution can be viewed as the sum of independent Bernoulli trials; since the number of trials is large in the case of large edge capacities, Chernoff-Hoeffding bounds [47] can be used to show that the probability that some edge capacity is violated is at most $1/m$ if μ is chosen as $\mu = e^{-1}4^{-1/\varepsilon}$. Furthermore, by the Markov inequality, the probability that the objective value resulting from the rounding is at least $\mu/2$ times the fractional optimum is at least $\mu/(2-\mu)$. Therefore, the probability that no edge capacity is violated and the objective value is at least $\mu/2$ times the fractional optimum is a constant. Thus, repeating the randomized rounding $\Theta(p)$ times provides a constant-factor approximation algorithm for the high-capacity case of MEDP with probability $1 - 2^{-p}$.

Theorem 7 (Raghavan and Thompson, 1987; Kleinberg, 1996). *For all instances of generalized* MEDP *where the edge capacities are at least* $\varepsilon \log m$ *for a suitable constant* ε*, there is a randomized polynomial-time algorithm that achieves a constant-factor approximation with high probability.*

Note that the constant-factor approximation ratio of this theorem holds also if the solution of the algorithm is compared with the *fractional* optimum of the LP relaxation of the problem.

Kolman and Scheideler [41] prove that a constant-factor approximation algorithm for generalized MEDP (in fact, even for the unsplittable flow problem) in graphs with flow number F exists if the edge capacities are at least $\varepsilon \log F$ for a suitable constant ε. Their result is obtained by a rather sophisticated use of randomized rounding. Note that for graphs with polylogarithmic flow number (hypercubes, expanders, etc.), this shows that constant approximation ratio can already be obtained if the edge capacities are $\Omega(\log \log n)$. This latter result had previously been obtained by Srinivasan [55] (see also Baveja and Srinivasan [5]).

Further results about the approximation ratio achieved by LP-based algorithms will be discussed in Section 5.5 in the context of the unsplittable flow problem.

3 Edge-Disjoint Paths in Specific Graph Classes

Since the approximation ratios that have been achieved for arbitrary graphs are quite large, it is interesting to investigate whether MEDP admits better approximation ratios for restricted classes of graphs. There are two basic techniques that sometimes allow to obtain a better approximation ratio on specific graph classes:

- For some graph classes, the approximation ratio of the standard greedy algorithm of Section 2.1 can be improved by sorting the given requests in some order depending on the structure of the given graph.
- For some graph classes, one can prove that there exists a solution using paths of length at most D that is at least as large as $1/\alpha$ times the optimum solution. The bounded-length greedy algorithm of Section 2.2 with parameter D and the shortest-path-first greedy algorithm of Section 2.3 both give approximation ratio at most αD in this case (cf. Kolman [38]).

We will see examples of both techniques in the following.

3.1 Trees and Trees of Rings

Recall that MEDP can be solved optimally in polynomial time for undirected trees, but is $\mathcal{APX}$-hard for bidirected trees and undirected or bidirected trees of rings. A simple approximation algorithm for MEDP in bidirected trees is the following: Root the given tree at an arbitrary node. For each node v of the tree, let $d(v)$ denote its distance from the root. For each request (s_i, t_i), let ℓ_i be the node on the path from s_i to t_i that is closest to the root. The node ℓ_i is also called the *lowest common ancestor* of s_i and t_i. Now process the given requests in order of non-increasing values $d(\ell_i)$ and apply the greedy algorithm (i.e., accept each path if it does not intersect any previously accepted path).

In order to analyze the algorithm, we fix an arbitrary optimal solution $\mathcal{A}^*$. When the algorithm accepts a request (s_i, t_i), we remove from $\mathcal{A}^*$ all paths that intersect (s_i, t_i). By definition of the algorithm, all paths $j \in \mathcal{A}^*$ have $d(\ell_j) \leq d(\ell_i)$. Therefore, if such a path $j \in \mathcal{A}^*$ intersects the path π_i from s_i to t_i, it must contain one of the (at most) two edges on π_i that are incident to ℓ_i.

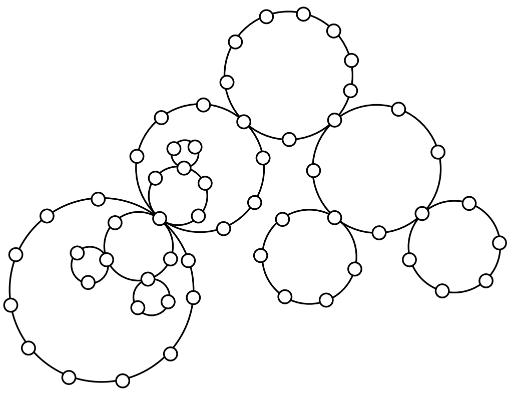

Fig. 6. A tree of rings

This means that there can be at most two paths in $\mathcal{A}^*$ that intersect π_i. Thus, the algorithm achieves approximation ratio 2.

An improved approximation algorithm with ratio $\frac{5}{3} + \varepsilon$, where $\varepsilon > 0$ is an arbitrary fixed constant, was presented by Erlebach and Jansen [21]. This algorithm is based on the simple 2-approximation just described, but it considers all paths with the same lowest common ancestor simultaneously and computes a maximum number s of such paths that can be added to the current solution. This can be done using a maximum bipartite matching algorithm. If $s \geq 3$, all these s paths are accepted. Since it suffices to remove from $\mathcal{A}^*$ at most s paths with the same lowest common ancestor and at most two other paths whose lowest common ancestor is closer to the root, at most $\frac{5}{3}s$ paths have to be removed from $\mathcal{A}^*$. If $s = 1$ or $s = 2$, the algorithm cannot always make the decision regarding acceptance or rejection right away. In some cases, it creates a configuration of *unresolved paths*, i.e., paths which are neither accepted nor rejected and whose status will be decided at a later node. Such configurations of unresolved paths complicate the processing at later nodes, but a careful case analysis shows that approximation ratio $\frac{5}{3} + \varepsilon$ can be achieved for any $\varepsilon > 0$. The running-time is polynomial in the size of the input for fixed $\varepsilon > 0$.

Theorem 8 (Erlebach and Jansen, 2001). *For every fixed* $\varepsilon > 0$*, there is a* $(\frac{5}{3} + \varepsilon)$*-approximation algorithm for* MEDP *in bidirected trees.*

An example of a tree of rings is shown in Fig. 6. In undirected or bidirected trees of rings, it is possible to reduce the problem to that for trees while losing at most a factor of 3 in the approximation ratio. The idea is just to remove one link from every ring of the tree of rings arbitrarily. This turns the tree of rings into a tree, and at least one third of the requests in the optimal solution for the tree of rings can still be routed in the tree. Thus, in the undirected case one can apply the optimal algorithm for undirected trees, leading to approximation ratio 3 for MEDP in undirected trees of rings. For bidirected trees of rings, approximation ratio $5+\varepsilon$ is obtained by using the $(\frac{5}{3}+\varepsilon)$-approximation algorithm for MEDP in bidirected trees of Theorem 8.

Theorem 9 (Erlebach, 2001). *There exists a* 3*-approximation algorithm for* MEDP *in undirected trees of rings and a* $(5+\varepsilon)$*-approximation algorithm for* MEDP *in bidirected trees of rings, for every fixed* $\varepsilon > 0$.

3.2 Meshes and Densely Embedded Graphs

Kleinberg and Tardos [36] found a randomized $O(1)$-approximation algorithm for MEDP in two-dimensional meshes. Furthermore, they generalized the algorithm to obtain approximation ratio $O(1)$ also for the class of *densely embedded, nearly-Eulerian graphs*; we refer to [36] for details. In the following, we give an outline of their algorithm for meshes.

Fix some constant γ. The given requests are partitioned into *long requests* and *short requests*. A request (s_i, t_i) is *short* if the distance between its endpoints is at most $16\gamma \log n$, and *long* otherwise. By considering short requests and long requests separately, computing an approximate solution for each of the two sets, and outputting the better of the two, an algorithm loses at most a factor of 2 in the approximation ratio. This is because at least half the requests in an optimal solution are either all short or all long.

The basic idea for dealing with the long requests is to embed a simulated network with high capacity into the mesh and to use an $O(1)$-approximation algorithm for high-capacity networks based on linear programming and randomized rounding (cf. Section 2.5) in this simulated network. One difficulty is translating the accepted paths in the simulated network into edge-disjoint paths in the original mesh.

The short requests are again partitioned into *medium requests* and *small requests*. The medium requests will be handled by applying the algorithm for long requests in separate submeshes of the original mesh. The small requests are short enough to be handled by brute-force enumeration.

Long Requests and the Simulated Network. For dealing with long requests, the mesh is partitioned into subsquares of size $\gamma \log n \times \gamma \log n$. The subsquare whose middle vertex is v is denoted by C_v. For a subsquare C_v, the up to eight other subsquares that are adjacent to it are called its *neighbors*. A maximal subset of the subsquares is chosen randomly such that no two chosen subsquares have a common neighbor. The random choice ensures that every subsquare becomes a chosen subsquare with constant probability and for any pair of subsquares whose middle vertices are at distance at least $11\gamma \log n$, there is a constant probability that both subsquares are chosen. The algorithm discards all requests that do not have both endpoints in chosen subsquares. This costs only a constant factor in the approximation ratio.

A chosen subsquare C_v together with all its neighbors becomes an *enclosure* D_v. Subsquares that do not belong to an enclosure at this point are included in any of their adjacent enclosures arbitrarily. Thus, the vertices of the mesh are

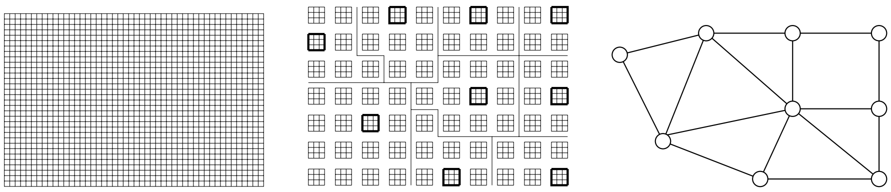

Fig. 7. The given mesh (left); its partitioning into $\gamma \log n \times \gamma \log n$ subsquares, with chosen subsquares drawn in bold and additional lines showing the border of the enclosures (middle); and the core of the resulting simulated network (right)

partitioned into enclosures. Two enclosures D_v and D_w are adjacent if there is an edge with one endpoint in D_v and the other endpoint in D_w. In this way, an enclosure can be adjacent to no more than 20 other enclosures. See Fig. 7 for an example.

Now the simulated network is built. First, take a vertex z_v for each chosen subsquare C_v, and connect two vertices z_v and z_w if their enclosures are adjacent. This defines the core $\mathcal{N}$ of the simulated network, and all edges in this core will be assigned capacity $\rho \log n$ for a constant $\rho < \gamma$. The right-hand side of Fig. 7 depicts the core of the simulated network in the example. To obtain the final simulated network $\mathcal{N}'$, add a copy of each chosen subsquare C_v and make all vertices at the boundary of C_v adjacent to z_v. The edges inside C_v and the edges from the boundary of C_v to z_v are assigned capacity 1.

The long requests with endpoints in chosen subsquares can be viewed as requests in $\mathcal{N}'$. Formulating the problem in $\mathcal{N}'$ as a linear program and solving this linear program, we obtain an optimal fractional solution F^* for the problem. The objective value of this fractional solution is within a constant factor of the optimal integral solution for the long requests in the mesh. By applying the randomized rounding approach of Raghavan and Thompson, we obtain an integral solution F (a set of accepted requests and one path for each accepted request) that, with high probability, does not violate any edge capacity in $\mathcal{N}$ and whose cardinality is a constant fraction of the fractional optimum. However, the paths in F may violate edge capacities in C_v.

The paths of the integral solution F in $\mathcal{N}'$ must be translated back into paths in the mesh. For each pair of adjacent enclosures D_v and D_w, choose a set $\tau_{v,w}$ of $\rho \log n$ edges connecting D_v and D_w arbitrarily. Furthermore, choose a set σ_v of $\rho \log n$ vertices equally spaced at the boundary of each C_v. The part of the mesh around a chosen subsquare C_v will be used to establish a crossbar structure allowing arbitrary interconnection of vertices in σ_v and endpoints of edges in all sets $\tau_{v,w}$ along edge-disjoint paths. The up to $\rho \log n$ paths going from z_v to z_w in $\mathcal{N}'$ will be routed through edges of $\tau_{v,w}$, and up to $\rho \log n$ paths with an endpoint s in C_v are to be routed from s to some vertex in σ_v.

To establish the crossbar structure, the $\frac{1}{2}\gamma \log n$ rings induced by vertices at distance $\gamma \log n$ to $\frac{3}{2}\gamma \log n$ from the middle vertex v of a chosen subsquare C_v are used. This is illustrated in Fig. 8. Each vertex in σ_v and each endpoint in D_v of an edge in some $\tau_{v,w}$ is assigned a different such ring; since D_v is adjacent

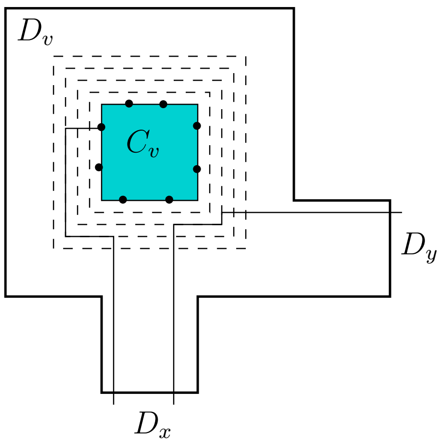

Fig. 8. The $\frac{1}{2}\gamma \log n$ rings outside C_v realize a crossbar structure in D_v. Two paths using the crossbar structure are shown: one path arrives from D_x and is routed to a vertex on the boundary of C_v, the other arrives from D_x and is routed to D_y.

to at most 20 other enclosures, this is possible if ρ is chosen smaller than $\gamma/42$. The outermost $\rho \log n$ rings are assigned to vertices in σ_v. It is easy to see that we can assign each vertex in σ_v and each endpoint in D_v of an edge in some $\tau_{v,w}$ a path from that vertex to its ring, such that all these paths are edge-disjoint and do not use an edge of any of the rings.

Consider a path π_F assigned to request (s_i, t_i) in F. Let s_i and t_i be contained in C_v and C_w, respectively. To route (s_i, t_i) in the mesh, a path is obtained as follows:

– Find a path from s_i to some vertex $r_{i,v}$ in σ_v and a path from t_i to some vertex $r_{i,w}$ in σ_w. These paths form the *first segment* and *last segment* of the constructed path from s_i to t_i in the mesh.
– If z_u is the first vertex after z_v in π_F, choose a free edge e_i in $\tau_{v,u}$, and let its endpoint in D_v be u_i. Follow the path from $r_{i,v}$ to its ring in D_v until it intersects the ring of u_i, then follow that ring and the path from that ring to u_i. Then take the edge e_i.
– Consider an intermediate vertex z_u with successor z_x on π_F. If the constructed path enters D_u through an edge e_1, pick a free edge e_2 in $\tau_{u,x}$. Assume without loss of generality that the ring of e_1 in D_u is inside the ring for e_2. Follow the path from e_1 to its ring until the ring of e_2 is hit, then follow the ring of e_2 and the path to e_2.
– At the last $\mathcal{N}$-vertex z_w of π_F (i.e., the last vertex of π_F that is in the core $\mathcal{N}$ of the simulated network $\mathcal{N}'$), if the partial path enters D_w through edge e_1, follow the path from e_1 to its ring in D_w until it intersects the ring of $r_{i,w}$, then follow that ring and its path to $r_{i,w}$.

Except for the first and last segment of the paths, it is clear that we find edge-disjoint paths in the mesh for each path in F.

It remains to deal with the first and last segments. Consider some subsquare C_v. The solution F can have more paths with an endpoint in C_v than can be routed along edge-disjoint paths to distinct vertices in σ_v. Kleinberg and Tardos model the *escape problem* of routing path endpoints in C_v to vertices in σ_v as a flow problem and observe that an edge-disjoint routing exists provided the cut condition is satisfied for all rectangular cuts. For the case that the cut condition is violated by the requests accepted in the solution F, they show that one can discard requests from F so that the remaining requests are a constant fraction of the requests in F and all cut conditions are satisfied. For the remaining requests, a network flow algorithm can be used to solve the escape problem and to obtain the first and last segments of the paths as required. In total, this gives an $O(1)$-approximation algorithm for the long requests.

Short Requests. Recall that the endpoints of short requests are at distance smaller than $16\gamma \log n$. Set $r = 32\gamma \log n$. First, the algorithm randomly chooses a maximal set M of vertices in the mesh with the property that the L_∞ distance between any two vertices in M is more than $4r$. For $u \in M$, let X_u be the set of all vertices with L_∞ distance at most r from u, and let Y_u be the set of all vertices with L_∞ distance at most $2r$ from u. Note that Y_u and $Y_{u'}$ are disjoint for $u, u' \in M$, $u \neq u'$. If M is chosen by an appropriate randomized algorithm, it can be shown that for any short request (s_i, t_i), there is a constant probability that both s_i and t_i are contained in X_u for some $u \in M$. Thus, the algorithm can restrict its attention to short requests with both endpoints in the same set X_u for some $u \in M$ and discard all other short requests. Furthermore, the optimal solution for the short requests with endpoints in X_u using paths in the submesh induced by Y_u can be shown to route at least one quarter of the requests in an optimal solution for these requests using paths in the whole mesh. Thus, the algorithm can deal with the requests in each X_u separately and use only the submesh induced by Y_u to route the requests with both endpoints in X_u.

Now consider the short requests with both endpoints in X_u. By considering these requests as requests in the submesh induced by Y_u, we can recursively partition them into long requests and short requests with respect to the submesh. Call the resulting long requests *medium requests* and the resulting short requests *small requests*. The medium requests are handled by the algorithm for long requests of the previous section, now applied to the submesh induced by Y_u. The small requests have endpoints whose distance is $O(\log \log n)$. By selecting subsets X'_v and Y'_v inside Y_u in the same way as X_u and Y_u were selected in the original mesh, the routing of small requests is reduced to solving MEDP in submeshes with side length $O(\log \log n)$. These submeshes are so small that an optimal solution can be computed by a brute-force enumeration in polynomial time.

Thus, we get approximation algorithms with constant ratio for the long requests, medium requests, and small requests. By running all three algorithms and outputting the largest of the three solutions, we get a constant-factor approximation algorithm for MEDP in meshes. Since some of the steps hold only with constant probability, parts of the algorithm must be repeated a certain number of times in order to guarantee that the constant-factor approximation is achieved with high probability.

Theorem 10 (Kleinberg and Tardos, 1995). *For* MEDP *in undirected meshes there is a randomized polynomial-time approximation algorithm that achieves a constant-factor approximation with high probability.*

Prior to this result, an $O(\log n \log\log n)$-approximation algorithm (that was an on-line algorithm) for meshes had been obtained by Awerbuch et al. [3] and $O(\log n)$-approximation algorithms by Aumann and Rabani [1] and Kleinberg and Tardos [35]. Kleinberg and Tardos [36] presented also an on-line algorithm with approximation ratio $O(\log n)$ for MEDP in meshes.

3.3 Hypercubes and de Bruijn Graphs

MEDP admits an $O(\log n)$-approximation algorithm in undirected or bidirected hypercubes. For bidirected hypercubes, Gu and Tamaki [29] have shown that any partial permutation can be realized with two sets of edge-disjoint paths. Taking the larger of the two sets gives a 2-approximation algorithm for MEDP on partial permutation instances. For general instances, we can first compute a largest subset of the given requests that is a partial permutation instance and then use the 2-approximation algorithm by Gu and Tamaki. By reducing a general instance to a partial permutation instance, we lose at most a factor of $O(\log n)$ in the approximation ratio, because the vertices of the hypercube have degree $d = \log n$. For undirected hypercubes, it was shown by Aumann and Rabani in [1] that any partial permutation can be realized by a constant number of sets of edge-disjoint paths. Hence, the same reasoning can be applied to undirected hypercubes as well. Since hypercubes have flow number $O(\log n)$, it follows from the work of Kolman and Scheideler [41] that *BoundedGreedy* achieves ratio $O(\log n)$ for MEDP in hypercubes. Hence, this approximation ratio can even be achieved in the on-line setting.

An embedding of the butterfly into the de Bruijn graph and a decomposition of the de Bruijn graph were presented by Kolman [38]. From these he derived, for every constant $\varepsilon > 0$, an $O(\log^{2+\varepsilon} n)$-approximation algorithm for MEDP in undirected de Bruijn graphs. As de Bruijn graphs have routing number $O(\log n)$ and constant maximum degree, it follows from the subsequent results by Kolman and Scheideler [40] on the routing number that *BoundedGreedy* is an $O(\log n)$-approximation algorithm for MEDP in de Bruijn graphs.

3.4 Expander Graphs

Kleinberg and Rubinfeld [34] considered MEDP in bounded-degree expander graphs. They fix a natural number $\Delta \geq 3$ and a real number $\alpha > 0$ arbitrarily, and then they assume that the maximum degree of the given graph G is at most Δ and that the expansion $\beta(G)$ is at least α.

Theorem 11 (Kleinberg and Rubinfeld, 1996). *For every constant $\varDelta$, the bounded-length greedy algorithm with parameter $D = c\varDelta \log n$ for a suitable constant c is an $O(\log n \log\log n)$-approximation algorithm for* MEDP *in undirected expander graphs with maximum degree at most $\varDelta$.*

To analyze the algorithm, they first show that there exists a feasible routing for a subset $\mathcal{R}'$ of $\mathcal{R}$ that is a partial permutation. Furthermore, $|\mathcal{R}'|$ is at least a constant fraction (more precisely, a $1/(\varDelta+1)$-fraction) of the optimal solution for $\mathcal{R}$. This follows, because each vertex is the endpoint of at most $\varDelta$ requests that are accepted in the optimal solution for $\mathcal{R}$, and so the requests in the optimal solution, viewed as edges between their endpoints, can be partitioned into $\varDelta + 1$ matchings by Vizing's edge coloring theorem. The largest matching gives the desired partial permutation $\mathcal{R}'$.

Next, they show that there is a set of requests $\mathcal{R}'' \subseteq \mathcal{R}'$ containing at least half of the paths in $\mathcal{R}'$ such that the requests in $\mathcal{R}''$ can be routed along paths of length at most $c \log n$ and at most $c' \log\log n$ paths are routed through the same edge. This result builds on previous work on routing of partial permutations in expanders by Broder, Frieze, and Upfal [12].

To analyze the bounded-length greedy algorithm, the same technique as in Section 2.2 can be used: Whenever the algorithm accepts a request and routes it along a path π, we remove that request and all other requests with paths that intersect π from $\mathcal{R}''$. We see that at most $|\pi| \cdot c' \log\log n + 1 = O(\log n \log\log n)$ requests have to be removed from $\mathcal{R}''$ in this way, thus proving the claimed approximation ratio.

Since expanders have routing number $O(\log n)$, it follows from the work of Kolman and Scheideler [40] that *BoundedGreedy* with parameter $D = O(\log n)$ achieves approximation ratio $O(\log n)$ for expanders, thus improving the result by Kleinberg and Rubinfeld.

Chakrabarti et al. [14] consider so-called *$\varDelta$-regular strong expanders* and show that approximation ratio $O(\sqrt{\log n})$ can be achieved for sufficiently large constant $\varDelta$.

3.5 Complete Graphs

For undirected or bidirected complete graphs, Erlebach and Vukadinović [22] have shown that there is a solution that uses only paths of length at most two and whose size is at least a constant fraction of the optimal solution. Therefore, the bounded-length greedy algorithm with $D = 2$ and the shortest-path-first greedy algorithm both achieve constant approximation ratio for MEDP in complete graphs. Using the results on the flow number due to Kolman and Scheideler [41], Carmi et al. [13] have shown that the approximation ratio achieved by the shortest-path-first greedy algorithm and the bounded-length greedy algorithm (with $D = 4$) is at most 9 and cannot be better than 3 for undirected complete graphs (which have flow number 1).

Chekuri and Khanna [15] prove that the shortest-path-first greedy algorithm has approximation ratio $O(n/\delta)$ for MEDP in undirected graphs with minimum degree δ. This also implies a constant-factor approximation algorithm for MEDP in complete graphs.

MEDP is $\mathcal{NP}$-hard in undirected and bidirected complete graphs [22], but no inapproximability result is known.

4 MEDP with Pre-determined Paths

In our definition of the MEDP problem, each request specifies only the endpoints s_i and t_i, and if the request is accepted, the path connecting s_i and t_i is determined by the algorithm. Alternatively, one can assume that every request with endpoints s_i and t_i already specifies a pre-determined path π_i connecting s_i and t_i. The path π_i might be determined by a separate routing algorithm or by the customer who submits the request. If request i is accepted, it must be routed along π_i. We denote this variant of MEDP by PreMEDP.

An instance I of PreMEDP with a set of k paths in a graph G can be viewed as a maximum independent set problem in the *conflict graph* of I, i.e., in the graph with one vertex for each of the k paths and with an edge between two vertices if the corresponding paths intersect.

Of course, the problems MEDP and PreMEDP are equivalent for trees. For undirected or bidirected rings, PreMEDP is polynomial, since the conflict graph is a circular-arc graph (or a disjoint union of two circular-arc graphs, in the bidirected case) in this case and the maximum independent set problem is polynomial for circular-arc graphs [30]. For undirected or bidirected trees of rings, PreMEDP was shown $\mathcal{APX}$-complete in [20]. For the approximation ratio achieved by a greedy algorithm based on depth-first search, upper bounds of 4 and 8 were given there for the undirected and bidirected case, respectively. For arbitrary graphs, it is easy to see that the natural adaptations of the bounded-length greedy algorithm and the shortest-path-first greedy algorithm achieve approximation ratio $O(\sqrt{m})$ for PreMEDP.

To get an inapproximability result for PreMEDP in arbitrary graphs, note that any graph H with n vertices can be encoded as the conflict graph of n paths in a series-parallel graph with $m = O(n^2)$ edges, such as the one shown

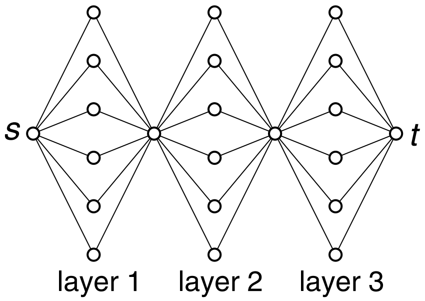

Fig. 9. A series-parallel graph

in Fig. 9: All n paths go from s to t. There are $O(n)$ layers between s and t, and each layer consists of $O(n)$ parallel chains of length two. The edge set of H can be split into $\ell = O(n)$ matchings M_1, M_2, …, M_ℓ using an edge coloring algorithm. In each layer i, the paths corresponding to the endpoints of the jth edge in M_i both use the jth chain of that layer. The paths corresponding to nodes that are not matched in M_i use one of the other chains of that layer (no two of them use the same chain). Thus, two of the constructed paths intersect if and only there is an edge between the corresponding vertices in H.

The maximum independent set problem in graphs with n vertices cannot be approximated within $n^{1-\varepsilon}$ for any $\varepsilon > 0$ unless $\mathcal{NP}$ = co-RP, as shown by Håstad [32]. Therefore, PreMEDP cannot be approximated within $m^{\frac{1}{2}-\varepsilon}$ for any $\varepsilon > 0$ under the same assumption. Unlike in the case of MEDP, this inapproximability result for PreMEDP applies also to undirected graphs and to graphs with bounded treewidth (since series-parallel graphs have treewidth at most two).

We remark that any graph with n vertices can also be encoded as the conflict graph of n paths in a mesh with $O(n^2)$ vertices, thus yielding the same inapproximability result. This construction was given by Nomikos in a reduction from the vertex coloring problem to path coloring with pre-determined paths [48].

For PreMEDP, one can also take the number of *rejected* paths as the objective value and turn the problem into a minimization problem, as suggested by Blum, Kalai and Kleinberg [6]. Viewed in the conflict graph, the goal is now to compute a minimum vertex cover. Since the minimum vertex cover problem can be approximated within a factor of 2 on arbitrary graphs, this shows that PreMEDP admits a 2-approximation for arbitrary graphs if the goal is to minimize the number of rejected paths.

5 The Unsplittable Flow Problem

The *unsplittable flow problem* (UFP) generalizes MEDP in several aspects. With respect to the given graph $G = (V, E)$, the difference is that every edge e now has a positive capacity $u(e)$. With respect to the set of requests $\mathcal{R}$, the difference is that in addition to specifying its endpoints s_i and t_i, each request i also has a positive *demand* d_i and a positive *profit* r_i. Unless stated otherwise, we assume that the edge capacities, demands, and profits can be arbitrary positive rational numbers. Again, a solution is given by selecting a subset of the given requests and assigning a path from s_i to t_i to each accepted request i. A solution is feasible if the edge capacity constraints are satisfied, i.e., if the sum of the demands of all accepted requests that are routed through an edge e is at most $u(e)$. The objective value of a solution is the sum of the profits of the accepted requests.

For a given instance of UFP, we let $u_{\min}$ and $u_{\max}$ denote the smallest and largest edge capacity, respectively. Similarly, $d_{\min}$ and $d_{\max}$ represent the smallest and largest demand of any request, and $r_{\min}$ and $r_{\max}$ are defined analogously. For a set $\mathcal{S}$ of requests, we define $r(\mathcal{S}) = \sum_{i \in \mathcal{S}} r_i$ and $d(\mathcal{S}) = \sum_{i \in \mathcal{S}} d_i$.

In addition to studying UFP in its full generality, many researchers have considered combinations of the following restrictions.

- The largest demand is at most the smallest edge capacity, i.e., $d_{\max} \le u_{\min}$. Since many results are known under this assumption, we call UFP with this restriction *classical* UFP. To distinguish the general UFP problem from classical UFP, it is also called *extended* UFP.
- We have $d_{\max} \le u_{\min}/K$ for some $K > 1$. This variant is called *bounded* UFP.
- All edge capacities are the same. This variant is called *uniform-capacity* UFP (UCUFP).
- All demands are the same.
- All profits are the same.
- The profit of a request is proportional to its demand.

In particular, MEDP is the special case of UFP in which $u(e) = 1$ for all $e \in E$ and $d_i = r_i = 1$ for all requests i. Therefore, all inapproximability results for MEDP apply also to UFP.

Even though UFP appears to be significantly more general than MEDP, an approximation ratio of $O(\sqrt{m})$ can be achieved for classical UFP as well.

5.1 UFP Approximations in Terms of Routing Number and Flow Number

Kolman and Scheideler [40] consider UCUFP in undirected graphs under the assumption that the profit of a request is equal to its demand. They suggest to use the bounded-length greedy algorithm with length parameter $D = 2R$, where R is the routing number of the given graph (cf. Section 2.2). If the algorithm processes the requests in order of non-increasing demands, they show that approximation ratio $O(\Delta R) = O(\Delta^2 \beta(G)^{-1} \log n)$ is achieved. They also present a randomized algorithm with approximation ratio $O(\Delta\sqrt{LR})$ if the graph has L-bounded routing number R.

In subsequent work, Kolman and Scheideler [41] obtain improved results by considering the *flow number* instead of the routing number. Let $G = (V, E)$ be an undirected graph with arbitrary positive edge capacities. For $v \in V$, let $u(v)$ be the sum of the capacities of the edges incident to v. Let $\Gamma = \sum_{v \in V} u(v)$. Consider a concurrent multicommodity flow problem I with a demand of $d_{v,w} = u(v)u(w)/\Gamma$ between every pair (v, w) of vertices in V. A feasible solution assigns to each pair (v, w) a (splittable) flow of at most $d_{v,w}$ units from v to w such that the total flow through each edge is at most the capacity of the edge. The *flow value* of a feasible solution is the maximum value f, $0 \le f \le 1$, such that at least $f \cdot d_{v,w}$ units of flow are routed for each pair (v, w). Finally, the *flow number* $F(G)$ is defined as the minimum, over all feasible solutions S to I, of $\max\{C(S), D(S)\}$, where $D(S)$ is the longest flow path used in S and $C(S)$ is the inverse of the flow value of S [41].

Kolman and Scheideler show that the flow number $F(G)$ of a graph can be computed in polynomial time and, using a result by Leighton and Rao [43], that $F(G) = \Omega(\beta(G)^{-1})$ and $F(G) = O(\Delta' \beta(G)^{-1} \log n)$ always hold, where $\Delta' = \max_{v \in V} u(v)$ and $\beta(G)$ is the expansion of G.

Then they prove that the bounded-length greedy algorithm with length parameter $D = 4F(G)$ achieves approximation ratio $16F(G) + 1 = O(F(G)) = O(\varDelta'\beta(G)^{-1}\log n)$ for the classical UFP in undirected graphs if the profit of a request is equal to its demand and if the algorithm processes the requests in order of non-increasing demand values. In terms of $\varDelta$ instead of $\varDelta'$, this gives approximation ratio $O(\varDelta\frac{u_{\max}}{u_{\min}}\beta(G)^{-1}\log n)$. For bounded UFP with $d_{\max} \le u_{\min}/K$ for some integer K, they present a modified algorithm with approximation ratio $O(K \cdot (F(G)^{1/K} - 1))$. If $K \ge \log F(G)$, this gives approximation ratio $O(\log F(G))$.

They also use an LP-based algorithm with randomized rounding to show that approximation ratio $O(1)$ can be achieved for classical UFP if $u_{\min}/d_{\max} \ge \gamma \log F(G)$ for a sufficiently large constant γ.

5.2 A Combinatorial Algorithm for Classical UFP

In the following, we present the combinatorial $O(\sqrt{m})$-approximation algorithm for classical UFP due to Azar and Regev [4]. First, the set of requests $\mathcal{R}$ is split into two subsets $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{R}_1$ contains all requests with $d_i \le u_{\min}/2$ and $\mathcal{R}_2$ all remaining requests. The algorithm computes a solution for each of the two sets separately and outputs the better of the two solutions. This costs at most a factor of 2 in the approximation ratio.

Consider one of the two sets R_q, $q = 1, 2$. The algorithm sorts the requests in order of non-increasing ratio r_i/d_i and processes them in this order. In this way, more "valuable" requests are processed first. For a request i and a path π from s_i to t_i, define

$$F(i,\pi) = \frac{r_i}{\sum_{e\in\pi}\frac{d_i}{u(e)}}.$$

Thus, $F(i,\pi)$ measures the profit gained relative to the added network load. The main idea of the algorithm is to fix a threshold α and to accept a request i only if there is a path π from s_i to t_i that has sufficient free capacity to accommodate the request and that satisfies

$$F(i,\pi) > \alpha. \tag{6}$$

The best choice for α is not known a priori. Therefore, the algorithm tries all powers of two in a certain range as possible values for α, computes a solution for each of these values, and outputs the best of them. A meaningful range for α is from $\alpha_{\min} = r_{\min}/n$ to $\alpha_{\max} = r_{\max}/(d_{\min}/u_{\max})$, because (6) is satisfied for all paths in case $\alpha \le \alpha_{\min}$ and for no path if $\alpha > \alpha_{\max}$. Thus, the algorithm tries out $\alpha = 2^j$ for $\lfloor \log \alpha_{\min} \rfloor \le j \le \lceil \log \alpha_{\max} \rceil$.

More precisely, the algorithm by Azar and Regev calls the subroutine *ThresholdGreedy* of Fig. 10 with each combination of parameters $\mathcal{S}$ and $\alpha = 2^j$ satisfying $\mathcal{S} = \mathcal{R}_1$ or $\mathcal{S} = \mathcal{R}_2$ and $\lfloor \log \alpha_{\min} \rfloor \le j \le \lceil \log \alpha_{\max} \rceil$. The best of the solutions obtained in this way is output.

To see that the algorithm has a polynomial running-time, note that the subroutine *ThresholdGreedy* is called

```
algorithm ThresholdGreedy($G = (V, E), \mathcal{S}, \alpha$):
  $\mathcal{A} \leftarrow \emptyset$;
  sort the requests in $\mathcal{S}$ in order of non-increasing $r_i/d_i$;
  for all requests $i$ in $\mathcal{S}$ in this order do
        if $\exists$ path $\pi_i$ from $s_i$ to $t_i$ in $G$ such that
        $F(i, \pi_i) > \alpha$ and every edge $e \in \pi$ has at least $d_i$ available capacity then
              $\mathcal{A} \leftarrow \mathcal{A} \cup \{(s_i, t_i)\}$;
              reduce the available capacity on all edges of $\pi_i$ by $d_i$;
        fi;
  od;
  return $\mathcal{A}$ and $\{\pi_i \mid (s_i, t_i) \in \mathcal{A}\}$;
```

Fig. 10. A threshold-based variant of the greedy algorithm

$$O\left(\log \frac{\alpha_{\max}}{\alpha_{\min}}\right) = O\left(\log \left(n \frac{r_{\max} u_{\max}}{r_{\min} d_{\min}} \right)\right)$$

times, so the number of calls is bounded by a polynomial in the size of the input. (Numbers in the input are encoded using a logarithmic number of bits.) Each call of *ThresholdGreedy* can be executed in polynomial time since the existence of a path π_i with enough available capacity and with $F(i, \pi_i) > \alpha$ can be checked using a shortest path algorithm, for the execution of which the weight of an edge e is taken to be $1/u(e)$.

Theorem 12 (Azar and Regev, 2001). *There is an $O(\sqrt{m})$-approximation algorithm for classical* UFP *in arbitrary directed or undirected graphs.*

Proof. Let $\mathcal{Q}$ be the set of requests in an optimal solution, and let π_i^* be the path assigned to request $i \in \mathcal{Q}$ by the optimal solution. Take $q = 1$ or $q = 2$ such that $\mathcal{Q} \cap \mathcal{R}_q$ contains at least half of the total profit of $\mathcal{Q}$. Let $\mathcal{Q}' = \mathcal{Q} \cap \mathcal{R}_q$. Note that $r(\mathcal{Q}') \geq r(\mathcal{Q})/2$.

Let $\alpha' = 2^{j'}$ be the largest value for α that is considered by the algorithm and that satisfies $r(\{i \in \mathcal{Q}' \mid F(i, \pi_i^*) > \alpha'\}) \geq r(\mathcal{Q}')/2$. It is clear that such an α' exists. We will show that *ThresholdGreedy*$(G, \mathcal{R}_q, \alpha')$ produces a solution $\mathcal{A}$ with $r(\mathcal{Q}')/r(\mathcal{A}) = O(\sqrt{m})$. Let π_i be the path assigned to request i in this set $\mathcal{A}$ by the algorithm.

Let $\mathcal{Q}'_{\text{high}} = \{i \in \mathcal{Q}' \mid F(i, \pi_i^*) > \alpha'\}$ and $\mathcal{Q}'_{\text{low}} = \{i \in \mathcal{Q}' \mid F(i, \pi_i^*) \leq 2\alpha'\}$ Note that $\mathcal{Q}'_{\text{high}}$ and $\mathcal{Q}'_{\text{low}}$ are not necessarily disjoint and each of them has total profit at least $r(\mathcal{Q}')/2$. We obtain

$$r(\mathcal{Q}'_{\text{low}}) = \sum_{i \in \mathcal{Q}'_{\text{low}}} F(i, \pi_i^*) \sum_{e \in \pi_i^*} \frac{d_i}{u(e)} \leq 2\alpha' \sum_{i \in \mathcal{Q}'_{\text{low}}} \sum_{e \in \pi_i^*} \frac{d_i}{u(e)} \leq 2\alpha' m\,,$$

since the solution $\mathcal{Q}'$ respects the edge capacities. Thus, we have $r(\mathcal{Q}') \leq 4m\alpha'$ and $r(\mathcal{Q}) \leq 2r(\mathcal{Q}') \leq 8m\alpha'$.

Call an edge e *heavy* if the total demand routed through the edge in the solution $\mathcal{A}$ is at least $u(e)/4$. Then define E_{heavy} to be the set of all heavy edges. We distinguish two cases.

Case 1. E_{heavy} contains at least $\sqrt{m}$ edges. Then we get

$$\begin{aligned} r(\mathcal{A}) &= \sum_{i\in\mathcal{A}} F(i,\pi_i) \sum_{e\in\pi_i} \frac{d_i}{u(e)} \geq \alpha' \sum_{i\in\mathcal{A}} \sum_{e\in\pi_i} \frac{d_i}{u(e)} \\ &\geq \alpha' |E_{\text{heavy}}|/4 \geq \alpha' \sqrt{m}/4. \end{aligned}$$

Thus we get $r(\mathcal{Q}) \leq 8m\alpha' \leq 32\sqrt{m} \cdot r(\mathcal{A})$.

Case 2. E_{heavy} contains less than $\sqrt{m}$ edges. In this case, we compare $r(\mathcal{A})$ with $r(\mathcal{Q}'_{\text{high}})$. Let $\mathcal{Q}'' = \mathcal{Q}'_{\text{high}} \setminus \mathcal{A}$. Since every request i in $\mathcal{Q}''$ is not accepted by the algorithm even though $F(i,\pi_i^*) > \alpha'$, this means that if the algorithm had routed request i along π_i^*, this would have exceeded the capacity of at least one of the edges on π_i^*, say, of the edge e_i.

We claim that e_i is a heavy edge. If $q = 1$, all requests in $\mathcal{R}_q$ have demand at most $u_{\min}/2$, thus the edge e_i can overflow only if it already carries a total demand of more than $u(e_i)/2$ and, consequently, is a heavy edge. If $q = 2$, all demands are between $u_{\min}/2$ and $u_{\min}$. If $u(e_i) \leq 2u_{\min}$, at least one request must already be routed through e_i and thus use at least $u_{\min}/2 \geq u(e_i)/4$ of the capacity of the edge. If $u(e_i) > 2u_{\min}$, a total demand of more than $u(e_i) - d_{\max} \geq u(e_i) - u_{\min} \geq u(e_i)/2$ must already be using the edge e_i. Again, we find that e_i is a heavy edge.

Let $\mathcal{Q}''(p)$ be the set of requests in $\mathcal{Q}''$ that are among the first p requests that are processed by the algorithm, and let $E(p) = \{e_i \mid i \in \mathcal{Q}''(p)\}$. Note that $E(p) \subseteq E_{\text{heavy}}$ and, therefore, $|E(p)| \leq \sqrt{m}$. Let $\mathcal{A}(p)$ be the requests that are accepted by the algorithm among the first p requests that it processes. Now the idea is to show that the total demand of $\mathcal{A}(p)$ is at least $\Omega(1/\sqrt{m})$ times the total demand of $\mathcal{Q}''(p)$. Since the requests are processed in order of non-increasing ratio r_j/d_j, we will then be able to conclude that $r(\mathcal{A})$ is $\Omega(1/\sqrt{m})$ times $r(\mathcal{Q}'')$.

Since each request in $\mathcal{Q}''(p)$ is routed through an edge of $E(p)$ in the optimal solution, we have $d(\mathcal{Q}''(p)) \leq \sum_{e\in E(p)} u(e)$. Let f be an edge in $E(p)$ with largest capacity. As f is heavy, we have $d(\mathcal{A}(p)) \geq u(f)/4$. So we get

$$d(\mathcal{Q}''(p)) < \sqrt{m} \cdot u(f) \leq 4\sqrt{m} \cdot d(\mathcal{A}(p)).$$

This implies that $r(\mathcal{Q}'') < 4\sqrt{m} \cdot r(\mathcal{A})$, since the requests are processed in order of non-increasing ratio r_i/d_i and the following claim can be verified by elementary calculations: Let $d_1, d_2, \ldots, d_\ell$ be a positive sequence and $b_1, b_2, \ldots, b_\ell$ a non-increasing positive sequence, and let $X, Y \subseteq \{1,\ldots,\ell\}$ and $X(p) = X \cap \{1,\ldots,p\}$ and $Y(p) = Y \cap \{1,\ldots,p\}$. If for every $1 \leq p \leq \ell$ we have $\sum_{j\in X(p)} d_j > \gamma \sum_{j\in Y(p)} d_j$, then

$$\sum_{j\in X} d_j b_j > \gamma \sum_{j\in Y} d_j b_j.$$

By taking $b_j = r_j/d_j$, $X(p) = \mathcal{A}(p)$, $Y(p) = \mathcal{Q}''(p)$, and $\gamma = 1/(4\sqrt{m})$, the above implication follows. From $r(\mathcal{Q}'') < 4\sqrt{m} \cdot r(\mathcal{A})$ and $r(\mathcal{Q}'_{\text{high}}) \leq r(\mathcal{Q}'') + r(\mathcal{A})$, we get $r(\mathcal{Q}'_{\text{high}}) \leq (4\sqrt{m} + 1)r(\mathcal{A})$ and thus $r(\mathcal{Q}) \leq 2r(\mathcal{Q}') \leq 4r(\mathcal{Q}'_{\text{high}}) \leq (16\sqrt{m} + 4)r(\mathcal{A})$.

In both cases, we have shown that $r(\mathcal{Q}) = O(\sqrt{m}) \cdot r(\mathcal{A})$. Thus, the algorithm achieves approximation ratio $O(\sqrt{m})$. $\square$

The running-time of the algorithm leading to Theorem 12 is polynomial in the size of the input, but it depends on the numbers that are part of the input, because *ThresholdGreedy* is called $O(\log\left(n\frac{r_{\max}u_{\max}}{r_{\min}d_{\min}}\right))$ times. Azar and Regev show that it is possible to modify the algorithm so that its running-time is *strongly polynomial*, i.e., bounded by a polynomial function of n, m, and k independent of the edge capacities, profit values, and demand values. To achieve this, first note that the capacity of an edge e with $u(e) > kd_{\max}$ can be reduced to $kd_{\max}$ without affecting the feasible solutions. Next, requests with profit below $r_{\max}/k$ can be discarded, while losing at most a factor of 2 in the approximation ratio. Then, the set $\mathcal{R}_{\text{tiny}}$ of requests with demand at most $u_{\min}/k$ are treated separately: the algorithm computes one solution consisting of all requests in $\mathcal{R}_{\text{tiny}}$ and one solution computed as before for the requests in $\mathcal{R} \setminus \mathcal{R}_{\text{tiny}}$. By outputting the better of the two solutions, again at most a factor of 2 is lost in the approximation ratio. In $\mathcal{R} \setminus \mathcal{R}_{\text{tiny}}$, all requests have demand at least $u_{\min}/k$. Thus, the ratio $r_{\max}u_{\max}/(r_{\min}d_{\min})$ is now bounded from above by $O(k^3)$, so the running-time of the algorithm is strongly polynomial.

Finally, we note here that Chekuri and Khanna [15] have extended their results for MEDP (cf. Section 2.3) to UCUFP: They gave algorithms with approximation ratio $O(\min\{\sqrt{m}, n^{2/3}\})$ for UCUFP in undirected graphs and $O(\min\{\sqrt{m}, n^{4/5}\})$ for UCUFP in directed graphs, under the assumption that the profit of each request is equal to 1. Their analysis works also if the profit of each request is equal to its demand.

5.3 Combinatorial Algorithms for Extended and Bounded UFP

Azar and Regev [4] present additional results for extended UFP and bounded UFP. For extended UFP, they obtain a combinatorial, strongly polynomial $O(\sqrt{m} \cdot \log(2 + \frac{d_{\max}}{u_{\min}}))$-approximation algorithm. The basic idea of the algorithm is to partition the given requests into $2 + \max\{\log \frac{d_{\max}}{u_{\min}}, 0\}$ classes depending on their demands and to run the algorithm of the previous section on each of the classes separately. In the end, the best of the computed solutions is output.

For bounded UFP, i.e., for the case of $d_{\max} \leq u_{\min}/K$ for some $K \geq 2$, Azar and Regev present a strongly polynomial $O(K \cdot D^{\frac{1}{K}})$-approximation algorithm, where D is the maximum possible length of a path assigned to a request.

For the extended UFP, Kolman and Scheideler [41] propose a variant of the bounded-length greedy algorithm that achieves approximation ratio $O(\sqrt{m})$ under the assumption that the profit of each request is equal to its demand. Kolman [39] generalizes the results due to Chekuri and Khanna [15] for UCUFP and shows that approximation ratio $O(\min\{\sqrt{m}, n^{2/3}\})$ and $O(\min\{\sqrt{m}, n^{4/5}\})$ can be achieved for extended UFP in undirected and directed graphs, respectively, provided that the profit of each request is equal to its demand.

5.4 Inapproximability Results for UFP

In Section 2.4, we have seen that there cannot be an approximation algorithm for MEDP in directed graphs that achieves approximation ratio $O(m^{\frac{1}{2}-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathcal{P} = \mathcal{NP}$. Since UFP is a generalization of MEDP, this inapproximability result applies to UFP as well. For the extended UFP, Azar and Regev present a stronger inapproximability result in [4]. They show that unless $\mathcal{P} = \mathcal{NP}$, no approximation algorithm for the extended UFP can achieve approximation ratio $O(m^{1-\varepsilon})$ for any $\varepsilon > 0$. (In terms of n, the result shows that no algorithm can have approximation ratio $O(n^{1-\varepsilon})$.) For instances of the extended UFP with $d_{\max}/u_{\min} \geq 2$ and any $\varepsilon > 0$, they prove that no approximation algorithm can have ratio better than $\Omega(m^{\frac{1}{2}-\varepsilon}\sqrt{\lfloor \log \frac{d_{\max}}{u_{\min}} \rfloor})$ unless $\mathcal{P} = \mathcal{NP}$. The proofs of these inapproximability results are again based on the $\mathcal{NP}$-completeness of problem 2DirPath and, hence, apply only to directed graphs.

5.5 LP-Based Algorithms for UFP

Prior to the work by Azar and Regev [4], a number of researchers had investigated the possibility of obtaining approximation algorithms for UFP from a linear programming relaxation. In Section 2.5, we have discussed the natural linear programming relaxation of MEDP. It is not difficult to see that the linear programming formulation of MEDP can be adapted to UFP in a straightforward way. In this context, the requests are often called *commodities* and the LP formulation is called a *fractional multicommodity flow problem.*

In case of extended UFP, it is important to ensure that a request (s_i, t_i) is routed only along paths π on which all edges have capacity at least d_i. This can easily be enforced in the LP formulation.

Throughout this section, we let OPT^* denote the objective value of the optimal solution to the LP-formulation and let OPT denote the integral optimum value. It is clear that $OPT \leq OPT^*$.

Srinivasan [55] presents approximation algorithms for UCUFP that are based on rounding the fractional solution of the LP relaxation. Without loss of generality, one can assume that $u(e) = 1$ for all $e \in E$. Under the assumption that the profit of each request is equal to its demand value, Srinivasan obtains a polynomial-time algorithm that outputs a feasible solution with total profit $\Omega(\max\{(OPT^*)^2/m, OPT^*/\sqrt{m}\})$. The bound $\Omega((OPT^*)^2/m)$ holds also in the case of arbitrary profit values in the interval $[0, 1]$.

Furthermore, Srinivasan considers fractional solutions to the LP with objective value at least $OPT^*/2$ such that each path that carries a positive fraction of a request (i.e., the paths π for which some variable $y_{i,\pi}$ is non-zero) consists of at most ℓ edges. He shows that in this case an $O(\ell)$-approximation is possible for UCUFP with arbitrary profits. Using results from [34], $\ell = O(\Delta^2\beta^{-2}\log^3 n)$ can always be achieved for UCUFP. Thus, for graphs in which Δ is bounded by a constant and β^{-1} is polylogarithmic, there is an approximation algorithm for UCUFP with polylogarithmic (in n) approximation ratio. This gives polylogarithmic approximation ratio for the butterfly and related hypercubic networks, which have $\Delta = O(1)$ and $\beta = \Theta(1/\log n)$. Srinivasan also shows that approximation ratio $O(\ell^{1/\varepsilon-1})$ can be achieved for UCUFP in the case where $d_{\max} \le 1-\varepsilon$ for some constant $\varepsilon \ge \frac{1}{2}$. This implies that for graphs like the butterfly, approximation ratio $O(1)$ can be achieved for UCUFP provided that $d_{\max} = O(1/\log\log n)$.

Kolliopoulos and Stein [37] obtain results concerning column-restricted packing integer programs and use them to derive LP-based algorithms for MEDP and classical UFP. For MEDP with arbitrary profit values, their algorithm computes a solution with total profit $\Omega(OPT^*/\sqrt{m})$ provided that the number k of requests is $O(m)$. For classical UFP with profit values in the interval $[0,1]$, their solutions have total profit $\Omega(OPT^*/(\sqrt{m}\log m))$, $\Omega((OPT^*)^2/(m\log^3 m))$ and $\Omega(OPT^*/\ell)$, where ℓ is defined as above.

Baveja and Srinivasan generalize the results of [55] to classical UFP in [5]. They show that it is possible to compute in polynomial time a solution with total profit at least $\Omega(\min\{OPT^*, (OPT^*)^2/m\})$ and $\Omega(OPT^*/\sqrt{m})$. The approximation algorithms with ratio $O(\ell)$ and $O(\ell^{1/\varepsilon-1})$ of [55] are also generalized to classical UFP and bounded UFP, respectively.

Guruswami et al. [31] consider LP-based algorithms that apply the randomized rounding technique [50] in a more direct way. They assume that all edge capacities and demand values are integral and that $d_{\max}$ is bounded by a polynomial in m. Their algorithms achieve approximation ratio $O(\sqrt{m}\log^{\frac{3}{2}} m)$ for extended UFP and $O(\sqrt{m\log m}\log\log m)$ for classical UFP. They also show that if $u_{\min}/d_{\max} \ge c\log m$ for a suitably large constant c, then UFP can be approximated within a constant factor by standard randomized rounding (cf. Section 2.5).

Chakrabarti et al. [14] give LP-based algorithms to obtain approximation ratio $O(\Delta\beta(G)^{-1}\log^2 n)$ for UFP in undirected graphs and $O(\Delta\beta(G)^{-1}\log n)$ for UCUFP in undirected graphs. They also show that the approximation ratios improve to $O((\Delta\beta(G)^{-1}\log^2 n)^{1/K})$ for UFP and $O((\Delta\beta(G)^{-1}\log n)^{1/K})$ for UCUFP in the bounded case with $d_{\max} \le u_{\min}/K$. Chakrabarti et al. [14] and Chekuri et al. [16] use LP-based algorithms also to obtain constant-factor approximation algorithms for classical UFP in chains, rings, and trees.

6 Further Results for Related Problems

In this survey we have focused on approximation algorithms for MEDP and UFP. Numerous results for closely related problems can be found in the literature. In this section we briefly mention some of them.

First, maximum path coloring (MAXPC) is the variant of MEDP where the input specifies an additional parameter $W \ge 1$ and the goal is to accept and route a subset of the requests such that the resulting paths can be partitioned into at most W sets of edge-disjoint paths. This problem is motivated by all-optical networks with wavelength-division multiplexing, because a network with W wavelengths can establish connections for W sets of edge-disjoint paths simultaneously. By a general reduction [17,2], a ρ-approximation algorithm for MEDP can be converted into an approximation algorithm with ratio at most $1/(1 - e^{-1/\rho}) \le \rho + 1$ for MAXPC. In some cases, better approximation ratios for MAXPC have been obtained using more direct approaches, for example by Nomikos et al. for MAXPC in undirected and bidirected rings [49].

Another problem related to MEDP is *path coloring*, where *all* given requests must be routed and assigned colors such that requests receive different colors if they share an edge. The goal is to minimize the number of colors used. Results for path coloring are surveyed in a different chapter of this book.

A variant of MEDP, called the bounded-length edge-disjoint paths problem, specifies a bound L on the lengths of the paths that the algorithm may assign to the requests. An $O(\sqrt{m})$-approximation for this problem is presented in [31]. That paper deals also with the integral-splittable flow problem, i.e., with the variant of UFP where the requests need not be routed along single paths, but can be split integrally among several paths (i.e., each demand is an integer and can be split among several paths in an integral way).

Instead of considering the maximization version of the edge-disjoint paths problem, one can also consider the question of how many terminal pairs in a graph can *always* be connected along edge-disjoint paths, no matter how the terminal pairs are chosen. For r-regular expander graphs, all sets of up to $\kappa = \Omega(n/\log n)$ pairs of vertices can be connected along edge-disjoint paths (provided no vertex appears as an endpoint of more than a constant number of requests), both in the undirected case [25] and in the directed case [9]. Since random graphs have good expansion properties with high probability, similar results could also be proved for random graphs of sufficiently high degree [11] and for random regular graphs [26].

While we have seen that UFP cannot be approximated within $O(m^{\frac{1}{2}-\varepsilon})$ unless $\mathcal{P} = \mathcal{NP}$, it has been shown that the single-source version of the problem admits constant-factor approximation algorithms [18,54]. In the single-source version, the vertex s_i of all requests (s_i, t_i) is the same.

Finally, we emphasize that we have mainly considered approximation algorithms for MEDP and UFP that have full knowledge about the input, i.e., that are off-line algorithms. In applications such as call admission control, it is meaningful to consider the on-line version of the problem, where the requests arrive over time and the algorithm must accept or reject each request without knowledge of future requests. The greedy algorithm and the bounded-length greedy algorithm that we discussed as approximation algorithms can in fact be applied as on-line algorithms, since they can process the requests in an arbitrary order. For surveys of known results on on-line MEDP and UFP, we refer the reader to Leonardi [44] and Borodin and El-Yaniv [10, Chapter 13]. More recent results can be found in Azar and Regev [4] and Kolman and Scheideler [41].

Acknowledgements

The author would like to thank Petr Kolman and an anonymous referee for numerous comments that helped to improve the presentation.

References

1. Y. Aumann and Y. Rabani. Improved bounds for all optical routing. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 567–576, 1995.
2. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. *Algorithmica*, 31(1):29–43, 2001.
3. B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS'94)*, pages 412–423, 1994.
4. Y. Azar and O. Regev. Strongly polynomial algorithms for the unsplittable flow problem. In *Proceedings of the 8th Integer Programming and Combinatorial Optimization Conference (IPCO)*, LNCS 2081, pages 15–29, 2001.
5. A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Mathematics of Operations Research*, 25(2):255–280, May 2000.
6. A. Blum, A. Kalai, and J. Kleinberg. Admission control to minimize rejections. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS 2001)*, LNCS 2125, pages 155–164, 2001.
7. H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.
8. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.
9. T. Bohman and A. Frieze. Arc-disjoint paths in expander digraphs. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 558–567, 2001.
10. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
11. A. Z. Broder, A. M. Frieze, S. Suen, and E. Upfal. Optimal construction of edge-disjoint paths in random graphs. *SIAM J. Comput.*, 28(2):541–573, April 1999.
12. A. Z. Broder, A. M. Frieze, and E. Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM J. Comput.*, 23(5):976–989, October 1994.
13. P. Carmi, T. Erlebach, and Y. Okamoto. Greedy edge-disjoint paths in complete graphs. In *Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'03)*, LNCS 2880, pages 143–155, 2003.
14. A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02)*, LNCS 2462, pages 51–66, 2002.
15. C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 628–637, 2003.

16. C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP'03)*, LNCS 2719, pages 410–425, 2003.
17. G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, April 1977.
18. Y. Dinitz, N. Garg, and M. X. Goemans. On the single source unsplittable flow problem. *Combinatorica*, pages 1–25, 1999.
19. T. Erlebach. *Scheduling Connections in Fast Networks.* PhD thesis, Technische Universität München, 1999.
20. T. Erlebach. Approximation algorithms and complexity results for path problems in trees of rings. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001)*, LNCS 2136, pages 351–362, 2001. See also TIK-Report 109, Computer Engineering and Networks Laboratory (TIK), ETH Zürich, June 2001.
21. T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14(3):326–355, 2001.
22. T. Erlebach and D. Vukadinović. New results for path problems in generalized stars, complete graphs, and brick wall graphs. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory (FCT 2001)*, LNCS 2138, pages 483–494, 2001.
23. S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.
24. A. Frank. Packing paths, circuits, and cuts – a survey. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows, and VLSI-Layout*, pages 47–100. Springer-Verlag, Berlin, 1990.
25. A. M. Frieze. Edge-disjoint paths in expander graphs. *SIAM J. Comput.*, 30:1790–1801, 2001.
26. A. M. Frieze and L. Zhao. Optimal construction of edge-disjoint paths in random regular graphs. *Combinatorics, Probability and Computing*, 9:241–264, 2000.
27. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York-San Francisco, 1979.
28. N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
29. Q.-P. Gu and H. Tamaki. Routing a permutation in the hypercube by two sets of edge disjoint paths. *Journal of Parallel and Distributed Computing*, 44(2):147–152, 1997.
30. U. I. Gupta, D. T. Lee, and J. Y.-T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12:459–467, 1982.
31. V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 19–28, 1999.
32. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 627–636, 1996.
33. J. Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.

34. J. Kleinberg and R. Rubinfeld. Short paths in expander graphs. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 86–95, 1996.
35. J. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 26–35, 1995.
36. J. Kleinberg and É. Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 52–61, 1995.
37. S. G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In *Proceedings of the 6th Integer Programming and Combinatorial Optimization Conference (IPCO VI)*, LNCS 1412, pages 153–168, 1998.
38. P. Kolman. Short disjoint paths on hypercubic graphs. Technical Report 2000-481, KAM-DIMATIA Series, Charles University, Prague, 2000.
39. P. Kolman. A note on the greedy algorithm for the unsplittable flow problem. Manuscript, December 2002.
40. P. Kolman and C. Scheideler. Simple on-line algorithms for the maximum disjoint paths problem. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'01)*, pages 38–47, 2001.
41. P. Kolman and C. Scheideler. Improved bounds for the unsplittable flow problem. In *Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 184–193, 2002.
42. M. E. Kramer and J. van Leeuwen. The complexity of wire routing and finding the minimum area layouts for arbitrary VLSI circuits. In F. P. Preparata, editor, *Advances in Computing Research; VLSI Theory*, volume 2, pages 129–146. JAI Press Inc., Greenwich, CT-London, 1984.
43. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, November 1999.
44. S. Leonardi. On-line network routing. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442. Springer-Verlag, Berlin, 1998.
45. B. Ma and L. Wang. On the inapproximability of disjoint paths and minimum Steiner forest with bandwidth constraints. *Journal of Computer and System Sciences*, 60:1–12, 2000.
46. M. Middendorf and F. Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, 13(1):97–107, 1993.
47. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
48. C. Nomikos. Approximability of the path coloring problem. In *ASHCOMP Workshop*, Udine, 1996.
49. C. Nomikos, A. Pagourtzis, and S. Zachos. Minimizing request blocking in all-optical rings. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, 2003.
50. P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
51. N. Robertson and P. Seymour. Graph Minors XIII: The disjoint paths problem. *Journal of Combinatorial Theory Series B*, 63:65–110, 1995.
52. C. Scheideler. *Universal Routing Strategies for Interconnection Networks.* LNCS 1390. Springer-Verlag, 1998.

53. A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Chichester, 1986.
54. M. Skutella. Approximating the single source unsplittable min-cost flow problem. *Mathematical Programming Series B*, 91(3):493–514, 2002.
55. A. Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS'97)*, pages 416–425, 1997.
56. K. Varadarajan and G. Venkataraman. Graph decomposition and a greedy algorithm for edge-disjoint paths. In *Proceedings of the 15th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 379–380, 2004.
57. J. Vygen. Disjoint paths. Technical Report 94816, Research Institute for Discrete Mathematics, University of Bonn, February 1994.
58. P.-J. Wan and L. Liu. Maximal throughput in wavelength-routed optical networks. In *Multichannel Optical Networks: Theory and Practice*, volume 46 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 15–26. AMS, 1998.
59. X. Zhou, S. Tamura, and T. Nishizeki. Finding edge-disjoint paths in partial k-trees. *Algorithmica*, 26:3–30, 2000.

Independence and Coloring Problems on Intersection Graphs of Disks

Thomas Erlebach[1,*] and Jiří Fiala[2,**]

[1] ETH Zürich, Computer Engineering and Networks Lab,
CH-8092 Zürich, Switzerland
erlebach@tik.ee.ethz.ch

[2] Charles University, Faculty of Mathematics and Physics,
Department of Applied Mathematics and
Institute for Theoretical Computer Science [***],
Malostranské nám. 2/25, 118 00 Prague, Czech Republic
fiala@kam.mff.cuni.cz

Abstract. This chapter surveys on-line and approximation algorithms for the maximum independent set and coloring problems on intersection graphs of disks. It includes a more detailed treatment of recent upper and lower bounds on the competitive ratio of on-line algorithms for coloring such graphs.

1 Introduction

The class of intersection graphs of disks in the Euclidean plane, called disk graphs, was studied for many years for its theoretical aspects as well as for its applications. As an example of a classical result we mention the theorem of Koebe who proved in 1936 that every planar graph can be represented as a coin graph, i.e. a disk graph where disks are not allowed to overlap [24] (see also the more accessible discussion of Koebe's result by Sachs [32]).

In contrast to the case of planar graphs, no efficient methods are known for the recognition of disk graphs. Breu and Kirkpatrick have shown that the recognition problem is *NP*-hard for unit disk graphs (intersection graphs of disks with equal diameter) [6] and for disk graphs with bounded diameter ratio (intersection graphs of disks where the ratio of the largest diameter to the smallest diameter is bounded by an arbitrary constant) [5]. Hliněný and Kratochvíl proved *NP*-hardness for the recognition problem of arbitrary disk graphs [17].

[*] Partially supported by EU Thematic Network APPOL II (IST-2001-32007), with funding provided by the Swiss Federal Office for Education and Science.

[**] This author was partially supported by EU APPOL project IST-1999-14084, during his postdoctoral fellowship at Christian-Albrechts University of Kiel and by NATO science fellowship, administered through Norwegian Research Council project no. 143192/410 during his postdoctoral fellowship at University of Bergen.

[***] Supported by the Ministry of Education of the Czech Republic as project LN00A056.

The hardness of the recognition problem implies that a disk representation cannot be derived from the graph in polynomial time unless $P = NP$. Therefore, an important factor in the design of algorithms for disk graphs is whether the disk graph is given only as a set of edges and vertices, or whether the centers and radii of the disks (called the *disk representation* of the graph) form the input to the algorithm. Some problems can be solved efficiently no matter whether the disk representation is given or not. The problem of computing a maximum clique in a unit disk graph is an example: Raghavan and Spinrad presented an efficient algorithm that does not require the disk representation [31].

The maximum independent set problem on disk graphs (computing a largest subset of the given disks such that the disks in the subset are pairwise disjoint) has applications in map labeling. Under the assumption that labels occupy a circular area, the maximum number of non-intersecting labels that can be placed on a map (out of a given set of desired labels) is equal to the size of the maximum independent set in the corresponding disk graph. For applications in map labeling, it is clearly interesting to extend the problem to rectangular labels (representing text), sliding labels, etc. For a bibliography on map labeling problems we refer to the on-line web catalogue maintained by Wolff and Strijk [34].

One of the most practical applications of disk graphs is in the channel assignment problem, where the aim is to assign frequencies to a collection of transmitters while avoiding interference. Hale pointed out in 1980 that the channel assignment problem can be modeled as a graph theoretical problem, if we assume that all transmitters have circular range and transmitters with intersecting ranges are to use different frequencies [15]. Clearly, the underlying graph is a disk graph, and the problem is equivalent to the graph coloring problem. Observe also that in this case we may assume that the disk representation can be derived from the placement of transmitters and their ranges.

We focus our study on approximation and on-line algorithms for the maximum independent set problem and the coloring problem for intersection graphs of disks. In Section 2, we survey results for the maximum independent set problem, while in Section 3, we treat the coloring problem. We consider general disk graphs as well as two subclasses: unit disk graphs and disk graphs with bounded diameter ratio. We provide known upper and lower bounds on the approximation and competitive ratios, and we also discuss the impact if the disk representation is given as part of the input. In Section 3.3, we give a more detailed account of recent results from [9] concerning on-line coloring of disk graphs and disk graphs with bounded diameter ratio.

1.1 Preliminaries

A *disk* in the Euclidean plane is specified by its center and its diameter. We denote the center of a disk D by c_D. Given a set of disks, the *intersection graph* of the disks is the graph with one vertex for each disk and with an edge between two vertices if the corresponding disks have a non-empty intersection.

Throughout this paper we consider only closed disks. Therefore, tangent disks are considered as intersecting. A graph G is called a *disk graph* if there exists a set of disks such that G is their intersection graph. Such a set of disks is called a *disk representation*, *disk model*, or *geometric representation* of G. A graph is called a *unit disk graph* if it is the intersection graph of a set of disks with the same diameter (w.l.o.g., we assume that the diameter is 1 in this case). The *diameter ratio* of a set of disks is the ratio of the maximum diameter of a disk to the smallest diameter. Intersection graphs of disks whose diameter ratio is bounded by σ are called *σ-bounded disk graphs.*

For a given graph $G = (V, E)$, the set of neighbors of a vertex $v \in V$ is denoted by $N(v) = \{u \in V : \{v, u\} \in E\}$. A subset $I \subseteq V$ is an *independent set* if the vertices in I are mutually non-adjacent. A subset $C \subseteq V$ is a *clique* if the vertices in C are pairwise adjacent. The maximum independent set problem and the maximum clique problem are the problems of computing a largest independent set and a largest clique, respectively. A *coloring* of G is an assignment of colors to vertices such that adjacent vertices receive different colors. The (minimum) coloring problem is the problem of computing a coloring using as few distinct colors as possible. These problems are notoriously hard to approximate on general graphs, but are often easier to approximate (or even to solve optimally) on restricted classes of graphs.

For an optimization problem, an *approximation algorithm* computes a feasible solution in time polynomial in the size of the input. It has *approximation ratio* ρ if for every input, the value of the computed solution is at most ρ times the optimum (for a minimization problem) or at least $1/\rho$ times the optimum (for a maximization problem). A *polynomial-time approximation scheme* (PTAS) is an algorithm that, given an instance of the problem and a parameter $\varepsilon > 0$, computes a feasible solution that is at most a factor of $1 + \varepsilon$ away from the optimum and whose running-time is polynomial in the size of the instance for every fixed $\varepsilon > 0$.

An on-line algorithm receives the vertices of the graph (or the disks) one by one together with the incident edges connecting the current vertex to previously presented vertices. It must decide the solution for the current vertex (membership in the independent set or the color assigned to the vertex) immediately without knowledge of future vertices and edges. An on-line algorithm achieves *competitive ratio* ρ if it always produces a solution that is at most a factor of ρ away from the optimum. Such an algorithm is called *ρ-competitive.*

2 The Maximum Independent Set Problem

The maximum independent set problem has been proved *NP*-complete for unit disk graphs even if the disk representation is given [33,8]. Of course, this implies that the problem is *NP*-complete for σ-bounded disk graphs and general disk graphs as well, and also in the case when the representation is not given. Therefore, one is interested in approximation algorithms for the problem.

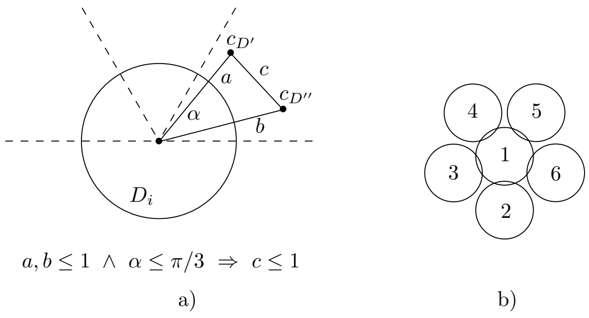
$c_{D'}$
a
c
$c_{D''}$
α
b
D_i
$a, b \leq 1 \ \wedge \ \alpha \leq \pi/3 \ \Rightarrow \ c \leq 1$
a)
4
5
1
3
6
2
b)

Fig. 1. Exploring the neighborhood of a unit disk

2.1 Independent Sets in Unit Disk Graphs

A first natural algorithm to consider is the greedy algorithm. It starts with an empty set $I = \emptyset$ and then processes the disks in arbitrary order. If the current disk is disjoint from all disks in I, it is added to I. When all disks have been processed, the set I is output. It is easy to see that the greedy algorithm does not require the disk representation and that it is an on-line algorithm.

Theorem 1 (Hochbaum [18], Marathe et al. [27]). *The greedy algorithm is a* 5*-competitive on-line algorithm for the maximum independent set problem in unit disk graphs. It does not require the disk representation.*

Proof. Consider some optimal solution I^*. Whenever the greedy algorithm accepts a disk D_i, remove from I^* all disks that intersect D_i (including D_i itself). It is clear that I^* is empty at the end of the greedy algorithm. Furthermore, we claim that each disk accepted by the greedy algorithm removes at most 5 disks from I^*, thus establishing that the competitive ratio is at most 5. Assume that the greedy algorithm accepts a disk D_i. If D_i is also contained in I^*, it suffices to remove one disk from I^*. If D_i is not contained in I^*, then all disks in I^* that intersect D_i have to be removed. There can be five such disks, as shown in Fig. 1 b). There cannot be six such disks, however: for any two disks D' and D'' that intersect D_i and that do not intersect each other, the angle between $\overline{c_{D_i} c_{D'}}$ and $\overline{c_{D_i} c_{D''}}$ must be larger than $\pi/3$, as illustrated in Fig. 1 a), and if there were six pairwise non-overlapping disks intersecting D_i, the sum of the angles between them would be larger than 2π, a contradiction. $\square$

Furthermore, no on-line algorithm can have competitive ratio better than 5 on unit disk graphs even if the disk representation is given: Assume that the input consists of the six disks in Fig. 1 b) and that the disk in the middle is presented first. Any on-line algorithm must accept the first disk (otherwise, it will have unbounded ratio on an instance consisting of this disk only) and will hence reject the other five disks, while the optimal independent set consists of these five disks.

As shown by Marathe et al. [27], the approximation ratio of the greedy algorithm can be improved from 5 to 3 in the off-line case by processing the given disks in order of non-decreasing y-coordinates of their centers. The argument in the proof of Theorem 1 can be adapted to show that among the disks that intersect the current disk D_i and that are processed later, there can be at most three pairwise disjoint disks, see again Fig. 1 a). Thus, at most three disks have to be removed from I^* for each disk accepted by the greedy algorithm.

This 3-approximation algorithm can be adapted to the case where the disk representation is not given as follows. A vertex whose neighborhood $N(v)$ does not contain an independent set of size larger than three can be found in polynomial time (e.g., in time $O(|V|^5)$ by enumerating all 4-element subsets of $N(v)$ for each $v \in V$). Such a vertex exists because the disk with lowest y-coordinate satisfies the property. Once such a vertex is determined, it is added to the independent set, and the vertex and all its neighbors are removed from the graph. This is repeated until the graph is empty.

Theorem 2 (Marathe et al. [27]). *There is a* 3*-approximation algorithm for the maximum independent set problem in unit disk graphs that does not require the disk representation.*

Using the method of local improvements, Halldórsson [16] showed that for every $\varepsilon > 0$ there is an algorithm achieving approximation ratio $\frac{5}{2} + \varepsilon$. This result was derived in a more general setting, for so called $(k+1)$-claw free graphs, i.e. graphs that do not contain $K_{1,k+1}$ as an induced subgraph. Unit disk graphs are 6-claw free as was already exhibited in the proof of Theorem 1. Halldórsson gave a $(\frac{k}{2} + \varepsilon)$-approximation algorithm for $(k+1)$-claw free graphs for any $k \geq 4$. The algorithm is based on local improvements using an analogue of augmenting paths of bounded length. The length of this path depends on the required precision ε, and therefore the exponent of the running-time grows if ε gets smaller.

Theorem 3 (Halldórsson [16]). *For every $\varepsilon > 0$, there is an approximation algorithm with ratio $\frac{k}{2} + \varepsilon$ for the maximum independent set problem in $(k+1)$-claw free graphs, and therefore a $(\frac{5}{2} + \varepsilon)$-approximation algorithm for unit disk graphs (not requiring the disk representation).*

For the case that the disk representation is given, a polynomial-time approximation scheme can be obtained using the shifting technique invented by Baker [3] and Hochbaum and Maass [19]. Such an approximation scheme was presented by Hunt III et al. [20] and by Matsui [28].

We sketch the basic ideas. Let a set $\mathcal{D}$ of disks be given. Denote by I^* some optimal independent set. Without loss of generality assume that the given disks have diameter 1 and that no center has an integral coordinate. Consider a grid consisting of horizontal and vertical lines at all integer coordinates. Fix some integer $k > 0$. For each pair of integers (i, j) such that $0 \leq i, j \leq k-1$, consider the subset $\mathcal{D}_{i,j}$ of disks obtained by removing all disks that intersect a vertical line at $x = i + kp$ for some $p \in \mathbb{Z}$ or some horizontal line at $y = j + kp$ for some

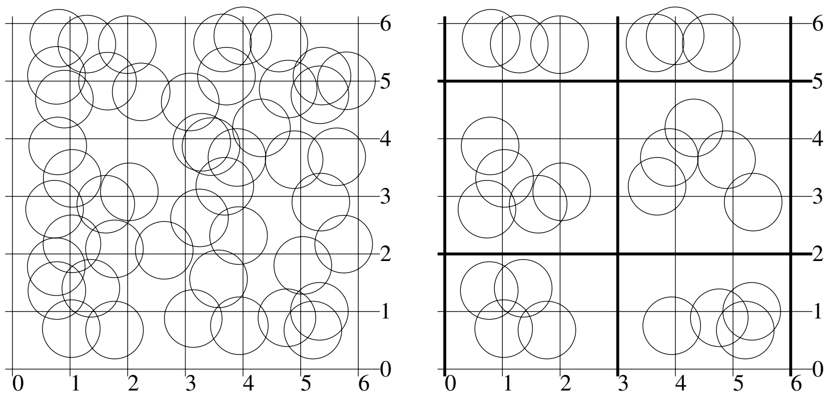

Fig. 2. Illustration of the shifting strategy for $k = 3$ and the choice $i = 0$ and $j = 2$. The given disks are shown on the left-hand side. If all disks intersecting a vertical line at $x = 3p$ for $p \in \mathbb{Z}$ or a horizontal line at $y = 2 + 3p$ for $p \in \mathbb{Z}$ (drawn in bold) are removed, the disks shown on the right-hand side remain and constitute $\mathcal{D}_{0,2}$.

$p \in \mathbb{Z}$. See Fig. 2 for an example. If the vertical lines at $x = i+kp$, $p \in \mathbb{Z}$, and the horizontal lines at $y = j + kp$, $p \in \mathbb{Z}$, are removed from the plane, disjoint open squares with side length k and area k^2 remain. Each disk in $\mathcal{D}_{i,j}$ is completely contained in one such square. Therefore, a maximum independent set of $\mathcal{D}_{i,j}$ is the union of maximum independents sets of disks in all squares. A maximum independent set among the disks in one square can contain at most $O(k^2)$ disks, and can hence be computed in time $|\mathcal{D}|^{O(k^2)}$ by enumeration of all subsets of size $O(k^2)$. Thus, a maximum independent set of $\mathcal{D}_{i,j}$ can be computed in polynomial time for fixed k. The algorithm computes a maximum independent set of $\mathcal{D}_{i,j}$ for all k^2 pairs (i, j) and outputs the largest among these sets as the solution. The approach is called *shifting* because trying all possible values of i and j can be viewed as shifting the grid through the plane.

The cardinality of the solution output by the algorithm is at least $(1-\frac{2}{k})|I^*|$. To see this, note that each disk intersects only one horizontal line at an integer coordinate and one vertical line, respectively. Hence, there exists a value of i such that at most $|I^*|/k$ disks in I^* intersect vertical lines $x = i + kp$ ($p \in \mathbb{Z}$). Similarly, there is a value of j such that at most $|I^*|/k$ disks in I^* intersect horizontal lines $y = j + kp$ ($p \in \mathbb{Z}$). Thus, the set $\mathcal{D}_{i,j}$ for these values of i and j still contains an independent set of size at least $(1 - \frac{2}{k})|I^*|$. Since the algorithm computes a maximum independent set in each $\mathcal{D}_{i,j}$, the largest such set must have cardinality at least $(1 - \frac{2}{k})|I^*|$.

For a given $\varepsilon > 0$, we can thus choose $k = \lceil 2/\varepsilon \rceil$ to obtain a solution of size at least $(1 - \varepsilon)|I^*|$. The running-time is $|\mathcal{D}|^{O(k^2)}$. It is shown in [20,28] that the running-time can be reduced to $|\mathcal{D}|^{O(k)}$ by removing only those disks that intersect a horizontal grid line at $y = j + kp$ for some $p \in \mathbb{Z}$ and then using dynamic programming to compute an optimal independent set in each strip of width k between two of these horizontal grid lines.

Theorem 4 (Hunt III et al. [20], Matsui [28]). *There is a polynomial-time approximation scheme for the maximum independent set problem in unit disk graphs provided that the disk representation is given as part of the input.*

Recently, by exploring the size of the maximum independent set in diameter-bounded disk graphs, a PTAS was derived also for the case when the disk representation is not known.

Theorem 5 (Nieberg et al. [29]). *The maximum independent set problem in unit disk graphs admits a polynomial-time approximation scheme even if the disk representation is not provided as part of the input.*

We now briefly explain the main idea of this approximation scheme. For a given graph G let $I_{v,i}$ stand for some maximum independent set in the subgraph of G induced by vertices at distance at most i from the vertex v of G.

It can be shown by geometric arguments that for an arbitrary unit disk graph $G = (V, E)$ and any fixed $\varepsilon > 0$, there exists a constant k with the following property: For any vertex $v \in V$ there exists a distance $k_v \leq k$ such that $|I_{v,k_v}| \leq (1 + \varepsilon)|I_{v,k_v-1}|$.

Therefore, for a vertex v one can identify a distance k_v with this property by computing $I_{v,i}$ for $i = 1, 2, \ldots, k_v$. As k_v is bounded by a constant, the sizes of the computed maximum independent sets are bounded by a constant as well. During this process, we can detect when i is equal to k_v, and at this moment we add I_{v,k_v-1} to the final independent set and remove from G all vertices at distance at most k_v from v. We repeat the above routine until all vertices of G have been removed and finally get the required $(1 + \varepsilon)$-approximation of the maximum independent set of G.

2.2 Independent Sets in General Disk Graphs

In general disk graphs, the greedy algorithm can have approximation ratio $n-1$ for instances with n disks: An instance could consist of one disk D (presented first) and $n-1$ smaller disks intersecting D, but not intersecting each other. The greedy algorithm would accept only the first disk, but the optimal solution would consist of the $n-1$ other disks. This instance also shows that no deterministic on-line algorithm can achieve competitive ratio better than $n-1$, even if the disk representation is given.

Theorem 6. *The greedy algorithm is a $(n-1)$-competitive algorithm for the maximum independent set problem in disk graphs. It works even if the disk representation is not given. No deterministic on-line algorithm can have competitive ratio smaller than $n-1$, even if the disk representation is given.*

In the off-line case, the idea of the 3-approximation algorithm for unit disk graphs of Theorem 2 can be adapted to give a 5-approximation algorithm for general disk graphs, as shown by Marathe et al. [27]. If the disk representation is given, it suffices to sort the disks in order of non-decreasing diameters and apply the greedy algorithm to this order. For each disk D_i, the set of disks that are processed later and that intersect D_i contains at most five pairwise disjoint disks. This follows by the same argument as in the proof of Theorem 2, because all disks that are processed later are at least as big as D_i. Thus, the greedy algorithm achieves approximation ratio 5 for disk graphs if it processes the disks in order of non-decreasing diameters. Again, the algorithm can be adapted to the case where the disk representation is not available: It suffices to identify a vertex whose neighborhood $N(v)$ does not contain an independent set of size at least 6. Such a vertex must exist, and it can be found in polynomial time.

Theorem 7 (Marathe et al. [27]). *There is a 5-approximation algorithm for the maximum independent set problem in disk graphs that does not require the disk representation.*

For the case that the disk representation is part of the input, a polynomial-time approximation scheme for the maximum independent set problem in disk graphs has been devised by Erlebach, Jansen and Seidel [10]. It is also based on the shifting technique, but the given disks are first partitioned into layers according to their diameters, and the shifting strategy is applied on all layers simultaneously, using grids of different granularity on different layers. For each choice of the shifting parameters i and j, $0 \leq i, j \leq k-1$, the optimal independent set in $\mathcal{D}_{i,j}$ is then computed using dynamic programming, beginning at the layer that contains the smallest disks. Independently, Chan [7] presented a polynomial-time approximation scheme based on similar ideas and using shifted quadtrees.

Theorem 8 (Erlebach, Jansen and Seidel [10], Chan [7]). *There is a polynomial-time approximation scheme for the maximum independent set problem in disk graphs provided that the disk representation is given as part of the input.*

It remains an open problem whether there is a polynomial-time approximation scheme for the maximum independent set problem in disk graphs also in the case where the representation is not given as part of the input. As far as we know, no inapproximability result for this case has appeared in the literature.

2.3 Independent Sets in Bounded Disk Graphs

Now consider σ-bounded disk graphs. Assume that the minimum diameter of a disk is equal to 1 and, consequently, the maximum diameter is at most σ. In the off-line case, we can just apply the algorithms for general disk graphs and get a 5-approximation algorithm for the case without given representation and a polynomial-time approximation scheme for the case with given representation.

We note here that for small values of σ the method of local improvements exhibited by Halldórsson [16] provides a better approximation ratio, since σ-bounded graphs are 10-claw free as far as $\sigma < \frac{1}{\sin(\pi/10)} - 1 \approx 2.236$. Moreover, the methods of Nieberg et al. [29] can be adapted to obtain a PTAS when the disk representation is not given. The only difference from the case of unit disk graphs is that the constant k now depends also on the diameter ratio σ.

In the on-line case, the competitive ratio of the greedy algorithm can be bounded by $O(\min\{\sigma^2, n\})$ for σ-bounded disk graphs. To see this, again let I^* denote some optimal independent set. When the algorithm accepts a disk D, we remove all neighbors of D (i.e., all disks intersecting D) from I^*. We claim that the neighborhood of D can contain at most $O(\sigma^2)$ disjoint disks. To see this, note that all neighbors of D must have their center within distance σ from the center of D. Thus, all neighbors of D are contained in a circle with radius 1.5σ around the center of D. This circle has area $2.25\sigma^2\pi$. Each disk has diameter at least 1 and thus occupies an area of at least $\pi/4$. This shows that the neighborhood of D can contain at most $9\sigma^2$ disjoint disks. So, for each disk accepted by the greedy algorithm, we have to remove at most $9\sigma^2$ disks from I^*. Thus, we get an upper bound of $O(\sigma^2)$ for the competitive ratio of the greedy algorithm. An upper bound of $n-1$ is trivial.

To show that no deterministic on-line algorithm can do better than the greedy algorithm in the worst case, consider an instance consisting of one disk with diameter σ followed by $\Theta(\sigma^2)$ disks of diameter 1 that intersect σ and that are disjoint from each other. On this instance, the solution of the greedy algorithm is a factor of $\Theta(\min\{n, \sigma^2\})$ smaller than the optimal solution.

Theorem 9. *The greedy algorithm achieves competitive ratio* $O(\min\{n, \sigma^2\})$ *for* σ*-bounded disk graphs with* n *disks. It does not require the disk representation. Every deterministic algorithm has competitive ratio* $\Omega(\min\{n, \sigma^2\})$ *even if the disk representation is given.*

3 The Coloring Problem

The problem of deciding whether a unit disk graph with given representation can be colored with three colors has been shown *NP*-complete by Clark, Colbourn and Johnson [8] using a reduction from 3-colorability of planar graphs with maximum degree 3. This implies that there cannot be an approximation algorithm for the coloring problem on unit disk graphs with approximation ratio smaller than 4/3.

As for the maximum independent set problem, the *NP*-completeness for unit disk graphs with given representation implies *NP*-completeness of deciding 3-colorability of disk graphs or σ-bounded disk graphs, and also for the variants without given representation.

It was proved by Gräf et al. in [13] that deciding k-colorability remains *NP*-complete for unit disk graphs for any fixed number $k \geq 3$.

3.1 Coloring Unit Disk Graphs

First-fit is one of the most well-known heuristics for on-line graph coloring. It processes the vertices of the given graph in some order and assigns each vertex the smallest available color, i.e., the smallest color that has not yet been assigned to an adjacent vertex. We assume that colors are represented by positive integers. If at most k vertices adjacent to v have been colored prior to v, the color assigned to v by First-fit is contained in $\{1, 2, \ldots, k+1\}$, because at most k of these colors can already have been assigned to neighbors of v.

Let us apply the First-fit coloring algorithm to unit disk graphs. Marathe et al. [27] proved an upper bound of 6 on the approximation ratio of this algorithm. We give an improved analysis leading to a bound of 5. Consider some disk D at the time it is assigned its color. Let $d(D)$ denote the number of intersecting disks that have been colored before. The color assigned to D is at most $d(D) + 1$. On the other hand, the closed neighborhood of D (the set of all disks intersecting D, including D itself) does not contain an independent set of size larger than 5. Thus, at most 5 of the disks in the closed neighborhood of D can be assigned the same color in any coloring. Therefore, even the optimal coloring must use at least $(d(D) + 1)/5$ colors. This shows that First-fit is a 5-approximation algorithm for coloring unit disk graphs. Furthermore, First-fit is an on-line algorithm that does not need the disk representation.

Theorem 10. *First-fit is an on-line algorithm with competitive ratio at most* 5 *for unit disk graphs. It does not require the disk representation.*

A lower bound of 2 on the competitive ratio of any on-line coloring algorithm for unit disk graphs was presented by Fiala et al. in [11]. We reproduce here the counterexample showing that no on-line algorithm can be $(2-\varepsilon)$-competitive for any $\varepsilon > 0$. Consider the graph and its representation depicted in Fig. 3 and order the disks as indicated by the numbers. Assume that there is an algorithm with competitive ratio $2 - \varepsilon$. The vertices 1–6 form an independent set and must be colored with the same color by the algorithm, because otherwise its competitive ratio would be at least 2 on these 6 vertices. On vertices 7–12 the algorithm may use two new colors (but no more, if its competitive ratio is to be smaller than 2). Then, however, it will need three extra colors for the central triple, thus using 6 colors in total. Since the optimal coloring uses only 3 colors, this contradicts the assumption that the algorithm is $(2 - \varepsilon)$-competitive.

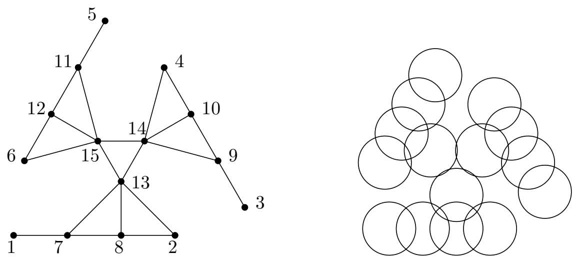

Fig. 3. A difficult instance for on-line unit disk coloring

Theorem 11 (Fiala et al. [11]). *No deterministic algorithm can achieve competitive ratio smaller than* 2 *for on-line coloring of unit disk graphs, even if the disk representation is part of the input.*

In the off-line case, 3-approximation algorithms for coloring unit disk graphs were presented by Peeters [30], Gräf et al. [13] and Marathe et al. [27]. If the disk representation is given, apply the First-fit algorithm to the disks in the order of non-increasing y-coordinates of their centers. Consider a disk D at the time it is assigned its color. The $d(D)$ previously colored disks that intersect D have a y-coordinate that is not smaller than the y-coordinate of D. As in the discussion leading to Theorem 2 (see also Fig. 1), the set of disks containing D and its previously colored neighbors does not have an independent set of size larger than 3. Therefore, even in an optimal coloring $(d(D) + 1)/3$ colors are required just for these disks. Since the color assigned to D is at most $d(D) + 1$, approximation ratio 3 is achieved.

To adapt the algorithm to the case without given representation, we can simply order the vertices in the recursively defined smallest-degree-last order and apply the First-fit algorithm. This means that we select some vertex v of minimum degree, remove v from the graph, color the resulting graph recursively, then insert v back into the graph and assign it the smallest available color. At the time v is removed (and thus also at the time it is inserted back again), the degree $d(v)$ of v is at most the degree $d(u)$ of the vertex u corresponding to a disk with the smallest y-coordinate. Thus, v is assigned color at most $d(v) + 1$, while the optimal coloring needs at least $(d(u)+1)/3 \geq (d(v)+1)/3$ colors. This shows that approximation ratio 3 is achieved.

Theorem 12 (Peeters [30], Gräf et al. [13], Marathe et al. [27]). *There is a* 3*-approximation algorithm for coloring unit disk graphs that does not require the disk representation.*

One can even show that the algorithm uses at most $3\omega(G) - 2$ colors, where $\omega(G)$ is the size of a maximum clique in the given unit disk graph. We sketch the idea for proving this in the case of given disk representation. When the algorithm colors a disk D, all previously colored neighbors have a y-coordinate not smaller than D. Thus, the previously colored neighbors can be covered by at most three cliques (see Fig. 1 a)): The disks with centers in the same region delimited by an angle of $\pi/3$ must form a clique (together with D). Thus, $d(D)$ is at most $3\omega(G) - 3$, so the algorithm uses at most $3\omega(G) - 2$ colors. In the case that the representation is not given, the smallest-degree-last First-fit algorithm uses at most $3\omega(G) - 2$ colors on unit disk graphs as well.

3.2 Off-Line Coloring of Disk Graphs

The approach for unit disk graphs was generalized to disk graphs by Marathe et al. [27]. They proved that the achieved approximation ratio for coloring disk graphs is at most 6, but their analysis can be sharpened to show that the smallest-degree-last First-fit algorithm is in fact a 5-approximation algorithm (cf. Gräf [12] and Malesińska [26]). To see this, note that the closed neighborhood of a disk with smallest diameter cannot contain an independent set of size 6 and hence such a disk has degree at most $5(\chi(G)-1)$, where $\chi(G)$ is the optimal number of colors (i.e., the chromatic number). Consequently, if a disk with the

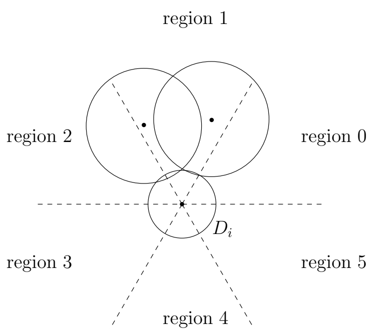

Fig. 4. The larger disks intersecting D_i can be partitioned into six cliques

smallest degree is colored last, this disk has at most $5(\chi(G) - 1)$ previously colored neighbors and is assigned color at most $5\chi(G) - 4$. Since the graph obtained after removing the disk of smallest degree is again a disk graph, the argument can be applied recursively. Thus, approximation ratio 5 is achieved. If the disk representation is given, we can alternatively sort the disks by non-increasing diameter and apply First-fit in this order to achieve the same bound.

Theorem 13 (Gräf [12]). *The smallest-degree-last First-fit algorithm achieves approximation ratio at most* 5 *for disk graphs. It does not need the disk representation.*

Furthermore, it can be shown that every disk graph G can be colored with at most $6\omega(G) - 6$ colors (Gräf [12] and Malesińska [26]). Consider the smallest disk D_i and the disks that intersect it. See Fig. 4. The plane around D_i can be partitioned into six regions delimited by an angle of $\pi/3$. Since all neighboring disks are at least as big as D_i, simple geometric arguments show that the neighboring disks whose centers are in the same region must form a clique. Furthermore, the regions can be chosen such that the center of at least one neighboring disk lies on the border between two regions, so that this neighboring disk belongs to the cliques of both regions. Therefore, the neighborhood of D_i can contain at most $6(\omega(G) - 1) - 1$ disks. Consequently, the algorithm that colors the smallest disk last needs at most $6\omega(G) - 6$ colors.

So far the only lower bound on the approximability of disk graph coloring is the one derived from the *NP*-completeness of deciding 3-colorability. It would be interesting to derive a larger lower bound that uses specific properties of disk graphs.

For σ-bounded disk graphs, no better approximation algorithms than for general disk graphs have been presented in the literature, although we expect that the upper bound on the approximation ratio achieved by the smallest-degree-last First-fit algorithm drops from 5 to 4 on disk graphs whose diameter ratio is bounded by a value σ that is smaller than some threshold (but still greater than 1).

On-line coloring of disk graphs and σ-bounded disk graphs is treated in detail in the next subsection.

3.3 On-Line Coloring of Disk Graphs

As a motivation for our further study of on-line coloring of disk graphs we shall mention the result of Gyárfás and Lehel from 1988 showing that there exists a tree T on n vertices such that for every on-line coloring algorithm there exists a specific ordering of the vertices of T, such that the algorithm is forced to use $\Omega(\log n)$ distinct colors [14]. Every tree is planar, so this result together with the theorem of Koebe immediately shows that this lower bound is valid also for disk graphs *without given representation*. In the following we will see that even the knowledge of the disk representation does not admit a better performance of a coloring algorithm. We give a proof that for every on-line disk coloring algorithm there exists a sequence of n disks such that the algorithm is forced to use at least $\Omega(\log n)$ distinct colors, while an optimal coloring uses only two colors. We also review a result of Irani [21] and explain how it can be used to prove that a competitive ratio of $O(\log n)$ is achieved by the First-fit coloring algorithm. This shows that the First-fit algorithm is optimal for on-line coloring of disk graphs up to a constant factor. Then, we consider σ-bounded disk graphs. For the case with given representation, we consider a different algorithm that achieves an improved competitive ratio of $O(\min\{\log n, \log \sigma\})$. For the case without representation, we get an upper bound of $O(\min\{\log n, \sigma^2\})$ on the competitive ratio of the First-fit algorithm.

A lower bound for on-line coloring of disks. Let $\mathcal{A}$ be an arbitrary on-line disk coloring algorithm. We prove that for any k there exists a sequence of disks $D_1, D_2, ..., D_n = \mathcal{D}(\mathcal{A}, k)$ (where $n = n(\mathcal{A}, k)$ depends on $\mathcal{A}$ and k, and is bounded from above by 2^k) such that:

- The intersection graph of $\mathcal{D}(\mathcal{A}, k)$ is isomorphic to a tree.
- The algorithm $\mathcal{A}$ is forced to use at least k distinct colors.

Let us start with some auxiliary notions: For a nonempty sequence of disks $\mathcal{D}$ we denote by $\mathcal{D}\flat$ the same sequence without the last element. We say that disks of some sequence are *in general position* if every pair of disks in the set differs in the maximum y-coordinate (i.e., the maximum y-coordinate of any point contained in the disk).

Now assume that the intersection graph of an arbitrary sequence of disks $D_1, ..., D_t$ in general position is a forest F. In each connected component of F we define the *active disk* to be the one with the highest maximum y-coordinate. The *active zone* of an active disk D_i is delimited by the interval spanned by the maximum y-coordinate of D_i and by the maximum y-coordinate of any other disk in the same connected component (if no other disk intersects D_i we use the mimimum y-coordinate of D_i).

The active zone of a sequence $\mathcal{D} = \{D_1, ..., D_t\}$ is defined as the intersection of the active zones of all active disks in $\mathcal{D}$. The width of an active zone is equal to the length of the corresponding interval, or is equal to 0 if the zone is empty.

Our construction of $D_1, ..., D_n = \mathcal{D}(\mathcal{A}, k)$ satisfies the following invariant:

Invariant 1. *For each on-line coloring algorithm $\mathcal{A}$, each $k \geq 2$, and any axis-aligned square B of side length ℓ, there exists a sequence $\mathcal{D}(\mathcal{A}, k)$ of at most 2^k disks such that*

1. *for every $D \in \mathcal{D}(\mathcal{A}, k)$, we have $D \subset B$, but no $D \in \mathcal{D}(\mathcal{A}, k)$ intersects the boundary of B,*
2. *the disks in $\mathcal{D}(\mathcal{A}, k)$ are in general position,*
3. *the intersection graph of $\mathcal{D}(\mathcal{A}, k)$ is a tree,*
4. *the active zones of $\mathcal{D}(\mathcal{A}, k)$ and $\mathcal{D}(\mathcal{A}, k)\flat$ have width at least $c^{-4^k}\ell$ for some constant c,*
5. *on the active disks of $\mathcal{D}(\mathcal{A}, k)\flat$ the algorithm $\mathcal{A}$ uses at least $k - 1$ distinct colors, and*
6. *the last disk in $\mathcal{D}(\mathcal{A}, k)$ intersects the active zones of all active disks of $\mathcal{D}(\mathcal{A}, k)\flat$ and is the active disk of $\mathcal{D}(\mathcal{A}, k)$.*

Observe that without loss of generality it is sufficient to prove the statement only for one fixed square B (e.g. for the unit square): Any algorithm $\mathcal{B}$ working on disks of some other square B' can be transformed into an algorithm $\mathcal{A}$ which colors a given disk $D \subset B$ by the same color as $\mathcal{B}$ colors $f(D)$, where f is a linear transformation of B to B'. Then the validity of the invariant for $\mathcal{A}$ on B implies that the invariant holds also for $\mathcal{B}$ on B'.

We present a construction of $\mathcal{D}(\mathcal{A}, k)$ satisfying the invariant by induction.

Our statement is clearly true for $k = 2$, when we select $\mathcal{D}(\mathcal{A}, 2)$ as two intersecting disks, e.g. placed at centers $(\frac{2}{5}, \frac{2}{5})$, $(\frac{3}{5}, \frac{3}{5})$ and of diameter $\frac{2}{5}$.

Now assume that the hypothesis is valid for k and we want to prove the statement for $k + 1$ and some bounding square B.

By the induction hypothesis there exists a set $\mathcal{D}^1 = \mathcal{D}(\mathcal{A}, k)$ such that $\mathcal{A}$ uses at least $k - 1$ distinct colors on the active disks of $\mathcal{D}^1\flat$ and all disks of $\mathcal{D}^1$ are inside the unit square.

The active zone of $\mathcal{D}^1\flat$ is of width at least c^{-4^k}. Now consider an axis-aligned square B' of side length c^{-4^k} placed in the active zone of $\mathcal{D}^1\flat$ such that it shares its left side with the right side of the bounding square of $\mathcal{D}^1$.

Denote by $\mathcal{B}$ the algorithm that behaves exactly as $\mathcal{A}$ after it colors $\mathcal{D}^1\flat$. Now by the hypothesis on $\mathcal{B}$ and B', there exists a sequence $\mathcal{D}(\mathcal{B}, k)$ such that $\mathcal{B}$ uses at least $k-1$ distinct colors on active disks of $\mathcal{D}(\mathcal{B}, k)\flat$, and as above this implies the existence of a set $\mathcal{D}^2$ satisfying the invariant for $\mathcal{A}$ and k. Moreover all disks of $\mathcal{D}^2$ are inside the active zone of $\mathcal{D}^1\flat$, but are disjoint from $\mathcal{D}^1$.

Now we distinguish the following two cases (see Fig. 5):

(1) The sets of $k - 1$ colors used on active disks of $\mathcal{D}^1\flat$ and $\mathcal{D}^2\flat$ are the same. Then the last disk of $\mathcal{D}^2$ uses the new k-th color and we place a new disk D_n that intersects the active disks of $\mathcal{D}^1\flat \cup \mathcal{D}^2$ but no other disk. We obtain $\mathcal{D}(\mathcal{A}, k + 1)$ as the concatenation of $\mathcal{D}^1\flat$, $\mathcal{D}^2$, and D_n.
(2) Active disks of $\mathcal{D}^1\flat$ and $\mathcal{D}^2\flat$ have different colors. This means that there appear at least k different colors, and we place a new disk D_n that intersects

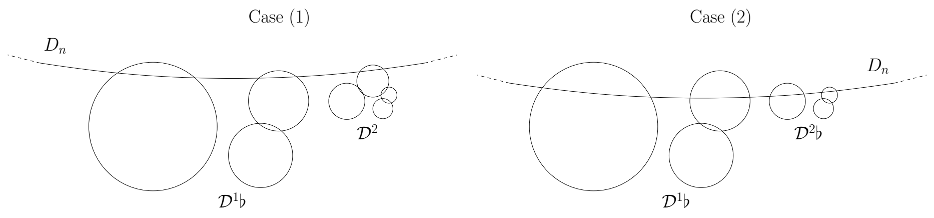

Fig. 5. Illustration of lower bound construction

only the active disks of $\mathcal{D}^1\flat\cup\mathcal{D}^2\flat$. We obtain $\mathcal{D}(\mathcal{A}, k+1)$ as the concatenation of $\mathcal{D}^1\flat$, $\mathcal{D}^2\flat$, and D_n.

Observe that all disks are in general position and the intersection graph is a tree. The only problem that might arise in the construction is that the diameter of the last disk D_n in $\mathcal{D}(\mathcal{A}, k+1)$ can be very large in order to ensure that this disk intersects the tiny active zones of all active disks in $\mathcal{D}(\mathcal{A}, k+1)\flat$, but no other disks.

However, by the construction, the active zone of $\mathcal{D}(\mathcal{A}, k+1)\flat$ has width at least $c^{-2\cdot 4^k}$. It suffices to find D_n among those disks which intersect the active disks in a strip of width at least $c^{-2\cdot 4^k}$ and length at most $1+c^{-4^k}$.

In this case, the diameter of the last disk D_n as well as the side length ℓ of the bounding square B are in $O(c^{2\cdot 4^k})$, so the width of the active zone of $\mathcal{D}(\mathcal{A}, k+1)\flat$ is at least $c^{-4^{k+1}}\ell$ for a suitable constant c.

Theorem 14 (Erlebach and Fiala [9]). *For every on-line disk coloring algorithm $\mathcal{A}$ there exists a sequence of n disks $\mathcal{D}$ such that $\mathcal{A}$ uses $\Omega(\log n)$ distinct colors on $\mathcal{D}$ while $\mathcal{D}$ can be colored optimally with two colors.*

Furthermore, by analyzing the diameter of the smallest disk used in the construction of $\mathcal{D}(\mathcal{A}, k)$, we can see that the diameter-ratio σ of $\mathcal{D}(\mathcal{A}, k)$ is bounded from above by $O(c^{4^k})$. Since $\mathcal{A}$ uses at least k colors on this instance, the competitive ratio of $\mathcal{A}$ must be $\Omega(\log\log\sigma)$.

Theorem 15 (Erlebach and Fiala [9]). *No deterministic on-line disk coloring algorithm can have competitive ratio $o(\log\log\sigma)$ on σ-bounded disk graphs.*

Upper bound for the First-fit algorithm. In order to analyze the competitive ratio of First-fit for on-line coloring of disks, we make use of a result due to Irani [21]. A graph G is called *d-inductive* if the vertices of G can be ordered in such a way that each vertex has at most d edges to higher-numbered vertices. Irani proved that if G is a d-inductive graph on n nodes, then First-fit uses $O(d\log n)$ colors to color G. In order to apply this result to disk graphs, we will show that these graphs are $O(\omega(G))$-inductive, where $\omega(G)$ denotes the size of a maximum clique in a graph G. First, we reformulate Irani's result in terms of the competitive ratio of First-fit on $O(\omega(G))$-inductive graphs.

Theorem 16 (Irani [21]). *Let d be a constant and let $\mathcal{G}$ be a class of graphs such that every $G \in \mathcal{G}$ is $d\omega(G)$-inductive. Then the First-fit coloring algorithm is $O(d \log n)$-competitive on graphs from $\mathcal{G}$, where n is the number of vertices of the given graph.*

Proof. We first introduce some terminology used later in the proof. Let G be a graph from the class $\mathcal{G}$. Let $v_1, v_2, \ldots, v_n$ be the order in which the vertices of G are colored. We write ω instead of $\omega(G)$. View each edge of G as being directed from the endpoint with smaller number to the endpoint with higher number in some vertex ordering that witnesses that G is $d\omega$-inductive. The fact that a directed edge goes from u to v is written as $u \succ v$. Then $\text{outdeg}(u) = |\{v : u \succ v\}|$. Note that $\text{outdeg}(u) \leq d\omega$ since G is $d\omega$-inductive. For a vertex v_j we define its set of successors as $S_j = \{v_i : v_j \succ v_i,\ i > j\}$. Observe that $|S_j| \leq d\omega$.

Assign to every vertex $v_j \in V(G)$ the value $C_j = 1$ and perform the following discharging procedure:

```
DISCHARGING PROCEDURE
for $j := 1$ to $n$ do
  begin
    $W_j := C_j$;
    if $S_j \neq \emptyset$ then
      for all $v_i \in S_j$ do $C_i := C_i + \frac{C_j}{|S_j|}$
    $C_j := 0$;
  end.
```

In every round of the outer for loop we get that

$$1 \leq W_j = C_j \leq \sum_{j=1}^{n} C_j \leq n.$$

We now bound W_j in terms of the maximum assigned color. Assume that a vertex v_j gets color c by the First-fit algorithm.

We prove by induction on c that $\frac{W_j}{|S_j|} \geq \frac{1}{d\omega}\left(1 + \frac{1}{d\omega}\right)^{c-1}$ whenever $S_j \neq \emptyset$, and $W_j \geq \left(1 + \frac{1}{d\omega}\right)^{c-1-d\omega}$ otherwise.

The statement is true for $c = 1$. For $c > 1$, there must be $c-1$ vertices v_i with $i < j$ that are adjacent to v_j and that have been assigned different colors. At most $d\omega - |S_j|$ of these $c-1$ vertices can come after v_j in the inductive order, so there must be at least $c - 1 - (d\omega - |S_j|)$ predecessors of v_j: $P_j = \{v_i : v_i \succ v_j,\ j > i\}$ that have been assigned different colors. Then, by the induction hypothesis:

$$\frac{W_j}{|S_j|} \geq \frac{1}{|S_j|}\left[1 + \sum_{v_i \in P_j} \frac{W_i}{|S_i|}\right] \geq \frac{1}{|S_j|}\left[1 + \frac{1}{d\omega} \sum_{i=1}^{c-1-d\omega+|S_j|} \left(1 + \frac{1}{d\omega}\right)^{i-1}\right] \geq$$
$$\geq \frac{1}{|S_j|}\left(1 + \frac{1}{d\omega}\right)^{c-1-d\omega+|S_j|}.$$

The last term is minimized in the case $|S_j| = d\omega$ and the claim follows. The case $S_j = \emptyset$ is discussed similarly.

We get that $n \geq \frac{1}{d\omega}(1 + \frac{1}{d\omega})^{c-1-d\omega}$ and therefore $c = O(d\omega \log n)$. □

Now we give bounds on the inductiveness of disk graphs in terms of $\omega(G)$.

Lemma 1. *Every disk graph G is $6\omega(G)$-inductive.*

Proof. Let $\mathcal{D}$ be a set of disks that is a disk representation of G. Order the disks in $\mathcal{D}$ according to non-decreasing diameters. Consider some disk D_i. By the arguments of the discussion after Theorem 13, the higher-numbered disks that intersect D_i can be partitioned into six groups such that the disks in each group form a clique. □

From Theorem 16 and Lemma 1 we get the following result.

Theorem 17 (Erlebach and Fiala [9]). *First-fit uses $O(\omega(G) \log n)$ colors to color a disk graph G with n nodes and is thus an $O(\log n)$-competitive on-line algorithm. It does not require the geometric representation.*

Coloring disks with bounded diameter ratio. We now focus our attention on the case that the ratio of the diameter of the largest disk and the smallest one is bounded by some value σ and that the disk representation is given as part of the input. (In fact, it would suffice that the diameters of the disks are given as part of the input.) We will see that there exists an on-line coloring algorithm with competitive ratio $O(\min\{\log n, \log \sigma\})$ in this case. The algorithm is a composition of two methods: The first method $\mathcal{A}$ is the First-fit technique for disk graphs with arbitrary diameter. It provides the bound $O(\log n)$. The second method $\mathcal{B}$ is the First-fit method applied separately on $O(\log \sigma)$ layers of disks, where the diameters of the disks on each layer are within a factor of two so that First-fit has constant competitive ratio on each layer.

More precisely, algorithm $\mathcal{B}$ assigns each disk D_i with diameter d_i to layer $\lfloor \log_2(d_i) \rfloor$ and applies First-Fit to each layer separately, using a different set of colors for each layer. The diameters of disks in the same layer differ at most by a factor of two, i.e., each layer is a 2-bounded disk graph.

Now it is not difficult to show that any disk D_i in a 2-bounded disk graph G can have at most $28\omega(G) - 6$ neighbors in G: This can be proved by dividing the plane around the center of D_i into 28 regions such that any two disks with centers in the same region must intersect; the detailed argument can be found in [9]. Therefore, the First-fit coloring algorithm uses at most $28\omega(G) - 5$ colors on G, and hence is 28-competitive for 2-bounded disk graphs.

Thus, Algorithm $\mathcal{B}$ achieves constant competitive ratio on each layer and, as the number of different layers is $O(\log \sigma)$, is an $O(\log \sigma)$-competitive coloring algorithm for σ-bounded disk graphs.

Now, it is possible to combine algorithms $\mathcal{A}$ and $\mathcal{B}$ as follows: We use two separate sets of colors for the algorithms $\mathcal{A}$ and $\mathcal{B}$. When a new disk D_i is presented we run $\mathcal{A}$ on D_i together with those disks colored by $\mathcal{A}$. Similarly, we execute $\mathcal{B}$. Then we compare the results of these two algorithms and color D_i with the algorithm that has used fewer colors up to now (including disk D_i). The total number of colors used on the entire set $\mathcal{D}$ is the sum of the number of colors used by both methods. Note that at any time of the execution of the combined algorithm, the number of colors used by $\mathcal{A}$ and the number of colors used by $\mathcal{B}$ differ by at most one.

Assume that $\log n < \log \sigma$. The number of colors used by algorithm $\mathcal{A}$ is at most $O(\log n)$ times the optimal number. The number of colors used by algorithm $\mathcal{B}$ is at most one more than that of $\mathcal{A}$. So the total number of colors used by the combined algorithm is $O(\log n)$ times the optimal number of colors. A symmetric argument holds in the case that $\log \sigma \leq \log n$. This yields the following theorem.

Theorem 18 (Erlebach and Fiala [9]). *If the disk representation is given as part of the input, there is an $O(\min\{\log n, \log \sigma\})$-competitive coloring algorithm for disk graphs whose diameter ratio is bounded by σ.*

Concerning on-line coloring of σ-bounded disk graphs in the case without given disk representation, First-fit is easily seen to be $O(\sigma^2)$-competitive. This follows because the neighborhood of a disk can be covered by $O(\sigma^2)$ cliques. The idea of the analysis is the same as the one outlined above for the case $\sigma = 2$. Combining this with the upper bound of $O(\log n)$, we get that First-fit is $O(\min\{\log n, \sigma^2\})$-competitive on σ-bounded disk graphs.

4 Conclusion

We have given a survey of known upper and lower bounds on the approximation ratio and competitive ratio achievable for the maximum independent set and minimum coloring problems on disk graphs. The bounds are summarized in Table 1.

For on-line coloring of disk graphs, we have shown that the First-fit method, which does not need the disk representation as part of the input, provides an $O(\log n)$-competitive disk coloring algorithm and that no algorithm can have competitive ratio $o(\log n)$, even if it uses the geometric representation. Furthermore, we showed that the geometric representation can help to get a better ratio of $O(\min\{\log n, \log \sigma\})$ on instances with diameter-ratio bounded by σ.

For this particular problem the gap between the lower bound $\Omega(\log \log \sigma)$ and the upper bound $O(\log \sigma)$ on the competitive ratio should be narrowed. Furthermore, we believe that the use of methods like randomization might improve the competitive ratio for the on-line problems studied here, and we expect further results in this direction.

The most widely used lower bound on the chromatic number of a disk graph (i.e., the lower bound on the optimal solution) is expressed via the clique number of the graph. We hope that by use of more sophisticated arguments it could be proven that standard coloring algorithms behave even better. As a particular open problem we would ask what is the supremum of the ratio of chromatic and clique number of a unit disk graph. The only known bounds are $1.5 \leq \frac{\chi(G)}{\omega(G)} < 3$.

Table 1. Summary of known results for disk graphs with n vertices. UDG stands for unit disk graphs, DG_σ for σ-bounded disk graphs, and DG for general disk graphs. PTAS stands for polynomial-time approximation scheme, NPC for *NP*-complete.

Problem	Graph class	Disk rep.	Approximation ratio		Competitive ratio	
			lower	upper	lower	upper
Maximum independent set	UDG	+	NPC	PTAS	5	
	DG_σ	+	NPC	PTAS	$\Theta(\min\{n, \sigma^2\})$	
	DG	+	NPC	PTAS	$n-1$	
	UDG	−	NPC	PTAS	5	
	DG_σ	−	NPC	PTAS	$\Theta(\min\{n, \sigma^2\})$	
	DG	−	NPC	5	$n-1$	
Minimum coloring	UDG	+	4/3	3	2	5
	DG_σ	+	4/3	5	$\Omega(\log\log\sigma)$	$O(\min\{\log n, \log\sigma\})$
	DG	+	4/3	5	$\Omega(\log n)$	$O(\log n)$
	UDG	−	4/3	3	2	5
	DG_σ	−	4/3	5	$\Omega(\log\log\sigma)$	$O(\min\{\log n, \sigma^2\})$
	DG	−	4/3	5	$\Omega(\log n)$	$O(\log n)$

The lower bound is derived from the coloring of the cycle C_5, and the upper bound is achieved by the algorithms due to Peeters [30], Gräf et al. [13] and Marathe et al. [27].

It should be mentioned that approximation algorithms for other *NP*-hard optimization problems besides maximum independent set and minimum coloring have also been studied for unit disk graphs and disk graphs. These problems include the weighted version of the maximum independent set problem, the minimum (weight) vertex cover problem, and the minimum dominating set problem (and variants thereof) [27,20,26,28,10]. Many of these results can be adapted to intersection graphs of arbitrary regular polygons, also in higher dimensions. One of the interesting open problems is to determine whether a polynomial-time approximation scheme exists for the minimum dominating set problem in general disk graphs with given representation. Another open problem is to settle the complexity of the maximum clique problem for general disk graphs (the problem is polynomial for unit disk graphs [8,31]). In this context it is interesting to note that the maximum clique problem in intersection graphs of ellipses has been proved *NP*-complete by Ambühl and Wagner [1].

Furthermore, a polynomial-time approximation scheme for the weighted fractional coloring problem on disk graphs was obtained by Jansen and Porkolab [22,23]. Previously, a 2-approximation algorithm for fractional coloring of unit disk graphs had been presented by Matsui [28].

Observe that several of the presented results for disk graphs hold analogously for intersection graphs of squares and for intersection graphs of axis-aligned squares. (The latter can be considered as disks in L_1 or L_∞ metrics.) On the other hand, we would like to point out that many questions are still open for ellipse and rectangle intersection graphs, a generalization of disk and square intersection graphs. The recognition problem for rectangle intersection graphs has been proved *NP*-complete by Kratochvíl [25]. For the maximum independent set problem in rectangle intersection graphs with n vertices, Berman et al. [4] presented a family of approximation algorithms achieving ratio $1 + \varepsilon \log n$ for any $\varepsilon > 0$. It is an open problem whether a constant approximation ratio can be obtained (even the existence of a PTAS cannot be excluded by the current state of knowledge). For coloring a rectangle intersection graph G with clique number $\omega(G)$, it is known that $O(\omega(G)^2)$ colors suffice [2], but no example with a non-linear lower bound in terms of the clique number has been obtained. For some special cases, it is known that $O(\omega(G))$ colors suffice [26]. It would be an interesting problem for future research to further investigate off-line and on-line coloring of ellipse and rectangle intersection graphs.

References

1. C. Ambühl and U. Wagner. On the clique problem in intersection graphs of ellipses. In *Proceedings of the 13th Annual International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science 2518, pages 489–500, 2002.
2. E. Asplund and B. Grünbaum. On a coloring problem. *Mathematica Scandinavica*, 8:181–188, 1960.
3. B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.
4. P. Berman, B. DasGupta, S. Muthukrishnan, and S. Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms SODA'01*, pages 427–436, 2001.
5. H. Breu and D. G. Kirkpatrick. On the complexity of recognizing intersection and touching graphs of disks. In *Proceedings of the Symposium on Graph Drawing (GD'95)*, Lecture Notes in Computer Science 1027, pages 88–98, 1996.
6. H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry: Theory and Applications*, 9(1–2):3–24, 1998.
7. T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46:178–189, 2003.
8. B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1–3):165–177, 1990.
9. T. Erlebach and J. Fiala. On-line coloring of geometric intersection graphs. *Computational Geometry: Theory and Applications*, 23(2):243–255, 2002.
10. T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms SODA'01*, pages 671–679, 2001.
11. J. Fiala, A. V. Fishkin, and F. V. Fomin. Off-line and on-line distance constrained labeling of graphs. In *Proceedings of the 9th European Symposim on Algorithms (ESA'01)*, Lecture Notes in Computer Science 2161, pages 464–475. Springer Verlag, 2001.
12. A. Gräf. Coloring and recognizing special graph classes. Musikinformatik und Medientechnik Bericht 20/95, Johannes Gutenberg-Universität Mainz, 1995.
13. A. Gräf, M. Stumpf, and G. Weißenfels. On coloring unit disk graphs. *Algorithmica*, 20(3):277–293, 1998. See also Musikinformatik und Medientechnik Bericht 17/94, Johannes Gutenberg-Universität Mainz, 1994.

14. A. Gyárfás and J. Lehel. On-line and first fit colourings of graphs. *Jornal of Graph Theory*, 12(2):217–227, 1988.
15. W. K. Hale. Frequency assignment: Theory and applications. *Proc. of the IEEE*, 68(12):1497–1514, 1980.
16. M. M. Halldórsson. Approximating discrete collections via local improvements. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 160–169, 1995.
17. P. Hliněný and J. Kratochvíl. Representing graphs by disks and balls. *Discrete Mathematics*, 229(1–3):101–124, 2001.
18. D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.
19. D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.
20. H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. NC-Approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *Journal of Algorithms*, 26(2):238–274, 1998.
21. S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11:53–72, 1994.
22. K. Jansen. Approximate strong separation with application in fractional graph coloring and preemptive scheduling. In *Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02)*, Lecture Notes in Computer Science 2285, pages 100–111, 2002.
23. K. Jansen and L. Porkolab. On preemptive resource constrained scheduling: Polynomial-time approximation schemes. In *Proceedings of the 9th Integer Programming and Combinatorial Optimization Conference (IPCO'02)*, Lecture Notes in Computer Science 2337, pages 329–349, 2002.
24. P. Koebe. Kontaktprobleme der konformen Abbildung. *Ber. Verh. Sächs. Akad. Leipzig*, 88:141–164, 1936.
25. J. Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Applied Mathematics*, 52:233–252, 1994.
26. E. Malesińska. *Graph theoretical models for frequency assignment problems.* PhD thesis, Technical University of Berlin, 1997.
27. M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.
28. T. Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Discrete and Computational Geometry*, Lecture Notes in Computer Science 1763, pages 194–200, 2000.
29. T. Nieberg, J. Hurink, and W. Kern. A robust PTAS for maximum weight independent sets in unit disk graphs. In *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'04)*, Lecture Notes in Computer Science 3353, pages 214–221, 2004.
30. R. Peeters. On coloring j-unit sphere graphs. Technical report, Dept. of Economics, Tilburg University, 1991.
31. V. Raghavan and J. Spinrad. Robust algorithms for restricted domains. In *Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 460–467, 2001.
32. H. Sachs. Coin graphs, polyhedra, and conformal mapping. *Discrete Mathematics*, 134:133–138, 1994.
33. D. Wang and Y.-S. Kuo. A study on two geometric location problems. *Information Processing Letters*, 28:281–286, 1988.
34. A. Wolff and T. Strijk. The map-labeling bibliography. http://i11www.ira.uka.de/map-labeling/bibliography/.

Approximation Algorithms for Min-Max and Max-Min Resource Sharing Problems, and Applications[*]

Klaus Jansen

Institut für Informatik und Praktische Mathematik,
Christian-Albrechts-Universität zu Kiel,
Olshausenstr. 40, 24098 Kiel, Germany
kj@informatik.uni-kiel.de

1 Introduction

In this paper we present a general technique to solve min-max and max-min resource sharing problems and show how to use it for two applications: scheduling jobs on unrelated machines, and strip packing. First we consider the following general type of min-max resource sharing problems. Given a non-empty convex set $B \subset \mathbb{R}^N$ and a set of M non-negative convex functions $f_i(x)$, the goal is to find a vector $x \in B$ that minimizes the value $\max\{f_1(x), \ldots, f_M(x)\}$. In the max-min resource sharing problem, the functions $f_i(x)$ are non-negative and concave and the goal is to find a point $x \in B$ that maximizes $\min\{f_1(x), \ldots, f_M(x)\}$. In both cases we assume to have a fast subroutine to solve a simpler optimization problem (called the corresponding block problem): to minimize or to maximize a non-negative convex or concave cost function over B, respectively. The subroutine is called the block solver. The feasibility variants to find a vector $x \in B$ such that $f(x) \leq e$ or $f(x) \geq e$, where e is the vector of all ones and the functions f_i are non-negative and linear, are called the fractional packing and covering problem, respectively.

Several different optimization problems can be described as min-max and max-min resource sharing problems [21,30]. Typical examples for min-max resource sharing and fractional packing problems are: scheduling jobs on unrelated machines, job shop scheduling, network embeddings, Held-Karp bound for TSP, multicommodity flows. In these applications, we have linear programming formulations with packing constraints (e.g. to place jobs on machines with an upper bound given by the schedule length). Applications for max-min resource sharing and fractional covering problems are: bin packing, strip packing, fractional graph colorings, scheduling parallel tasks and resource constrained scheduling. In these applications, the used linear programming relaxations contain mainly covering constraints (e.g. to cover the processing times of jobs).

We focus on computing approximate solutions for min-max and max-min resource sharing problems and on two applications. For a given relative tolerance

[*] Supported by EU Thematic Network APPOL I+II, Approximation and Online Algorithms, IST-1999-14084 and IST-2001-30012.

$\epsilon > 0$, a point $x \in B$ is an ϵ-approximate solution for the min-max resource sharing problem if $f_i(x) \leq (1+\epsilon)\lambda^*_{\mathcal{P}}$ where $\lambda^*_{\mathcal{P}}$ is the optimum objective value (i.e. $\lambda^*_{\mathcal{P}} = \min\{\lambda | f_i(x) \leq \lambda, x \in B, i = 1, \ldots, M\}$). A point $x \in B$ is an ϵ-approximate solution for the max-min resource sharing problem if $f_i(x) \geq (1-\epsilon)\lambda^*_{\mathcal{C}}$ where $\lambda^*_{\mathcal{C}} = \max\{\lambda | f_i(x) \geq \lambda, x \in B, i = 1, \ldots, M\}$). The running times of the algorithms that we describe (proposed by Grigoriadis et al. [15,16]) depend polynomial on $1/\epsilon$ and the number of constraints M. The algorithms are based on a Lagrangian or price-directive decomposition method and compute a sequence of vectors $x \in B$ to approximate the optimum solution. One Lagrangian decomposition step consists of three substeps:

(1) using the current $x \in B$, the algorithm computes a price vector $p(f(x)) = (p_1(f(x)), \ldots, p_M(f(x)))$,
(2) it computes a solution $\hat{x} \in B$ of the block problem with price vector $p(f(x))$,
(3) the algorithm moves from x to $(1-\tau)x + \tau\hat{x}$ with an appropriate step length $\tau \in (0,1]$.

We call each such Lagrangian decomposition iteration a coordination step. The total running time of the algorithm can be bounded mainly by the number of iterations and the running time of the block solver. Interestingly, the number of iterations or coordination steps is independent on the number of variables N.

Plotkin et al. [30] considered the linear feasibility variants of both problems: either to find a point $x \in B$ such that $f(x) \geq (1-\epsilon)e$ or to find a point $x \in B$ such that $f(x) \leq (1+\epsilon)e$ where e is the vector of all ones. The problems are solved in [30] by Lagrangian decomposition using exponential potential functions. Up to poly-logarithmic factors, the number of iterations (calls to the corresponding block solver) in these algorithms is proportional to M, $1/\epsilon^2$ and ρ, where $\rho = \max_{1 \leq i \leq M} \max_{x \in B} f_i(x)$ is the width of B relative to $f(x)$. The problems are further decomposed in [30] in order to reduce this linear dependence on ρ downto $\log \rho$. However, this introduces additional constraints on the block problems, which in general become NP-hard or harder to approximate.

In this paper we describe the framework based on logarithmic potential functions [15,16] for both variants: the min-max and max-min resource sharing problem in the general form (i.e. with convex and concave functions). The potential functions are used in the algorithms to compute the price vector $p(f(x))$. In Section 2, we give a width-independent Lagrangian decomposition iteration bound for the min-max resource sharing problem. It is based on an algorithm for the special case with starting solution $\lambda_{\mathcal{P}}(f(x_0)) = \max\{f_m(x_0) | m = 1, \ldots, M\} \leq 1$. In the general case, the functions are modified by scaling and the algorithm is based on a ternary search procedure. The number of iterations (calls to the block solver) for the min-max resource sharing problem is $O(M(\epsilon^{-2} \ln \epsilon^{-1} + \ln M))$. Garg and Könemann [14] proposed an algorithm (using an exponential potential reduction) for the fractional packing problem that needs $O(M\epsilon^{-2} \ln M)$ iterations. Recently, Jansen and Zhang [22] have proposed an improved algorithm for the min-max resource sharing problem with $O(M(\epsilon^{-2} + \ln M))$ coordination steps. In Section 3, we describe a width-independent Lagrangian iteration bound for the max-min resource sharing problem again using a logarithmic potential function. The number of iterations (calls to the block solver) for the max-min resource sharing problem is $O(M(\epsilon^{-2} + \ln M))$.

Afterwards we describe two applications in detail. In Section 4 we discuss a classical scheduling problem: scheduling jobs on unrelated machines. Given n jobs with execution times p_{ij} (that depend on the machines and jobs) and m machines, the goal is to find a schedule for the jobs that minimizes the schedule length (i.e. the maximum completion time). We describe a 2 - approximation algorithm proposed by Lenstra, Shmoys and Tardos [28]. The algorithm is based on a linear programming approach and rounding of fractional variables. The main bottleneck in this algorithm is to solve a linear program with packing constraints. Using the min-max resource sharing framework we show how to improve the running time of this step to get a faster $(2 + \epsilon)$ approximation algorithm.

In Section 5 we discuss another problem as an application of the max-min resource sharing approach. The problem is that of packing rectangles into a strip (called strip packing). The goal is to find a packing of the rectangles in the strip with minimum height. This problem is related to that of scheduling parallel multiprocessor tasks. We describe an asymptotic fully polynomial time approximation scheme for strip packing (for the case with bounded heights $h_i \leq 1$ of the rectangles) proposed by Kenyon and Remila [25]. Here the bottleneck in the computation is the solution of a huge linear program (a fractional strip packing problem) with exponential number of variables. The linear programming relaxation used contains covering constraints. We show how to solve the fractional strip packing problem using the max-min resource sharing framework to get a faster asymptotic fully polynomial time approximation scheme.

In Section 6 and 7 we give the details for the analysis of the algorithms by Grigoriadis and Khachiyan [15,16] described in Sections 2 and 3. For further reading on min-max and max-min resource sharing problems we refer the reader to [4,14,15,16,30,36] and for more information on scheduling problems we refer the reader to [5,6,17].

2 Min-Max Resource Sharing

In this section we discuss an algorithm for the min-max resource sharing problem proposed by Grigoriadis and Khachiyan [15]. We consider the following optimization problem $(\mathcal{P})$:

$$\begin{aligned} &\text{Min } \lambda \\ &\text{s.t. } f_i(x_1, \ldots, x_N) \leq \lambda, \qquad i = 1, \ldots, M, \\ &\qquad (x_1, \ldots, x_N) \in B, \end{aligned}$$

where $B \subset \mathbb{R}^N$ is a nonempty convex compact set and $f_i : B \to \mathbb{R}^+$ are non-negative continuous convex functions on B for $i = 1, \ldots, M$.

Let $f(x) = (f_1(x), \ldots, f_M(x))^T$ for $x = (x_1, \ldots, x_N)^T \in B$, and let

$$\lambda^*_{\mathcal{P}} = \min\{\lambda \mid f(x) \leq \lambda e, x \in B\}$$

where $e = (1, \ldots, 1)^T$. We are interested to compute an ϵ - approximate solution for $(\mathcal{P})$ for a given relative tolerance $\epsilon \in (0, 1)$:

$$(\mathcal{P}_\epsilon) \qquad \text{compute } \ x \in B \ \text{ such that } \ f(x) \ \leq \ (1+\epsilon)\lambda^*_{\mathcal{P}} e.$$

From the von Neumann's saddlepoint theorem we have the following duality relation:

$$\lambda^*_{\mathcal{P}} = \min_{x \in B} \max_{p \in P} p^T f(x) = \max_{p \in P} \min_{x \in B} p^T f(x),$$

where $P = \{p \in \mathbb{R}^M | \sum_{i=1}^M p_i = 1, p_i \geq 0\}$. Let $\Lambda(p) = \min\{p^T f(x) | x \in B\}$ be the corresponding block problem associated with $\mathcal{P}$. Then $\lambda^*_{\mathcal{P}} = \max\{\Lambda(p) | p \in P\}$. Based on the fact that $\lambda^*_{\mathcal{P}}$ is the optimum value of the Lagrangian dual, we define the approximate dual problem as:

$$(\mathcal{D}_\epsilon) \qquad \text{compute } \ p \in P \ \text{ such that } \ \Lambda(p) \ \geq \ (1-\epsilon)\lambda^*_{\mathcal{P}}.$$

We use here the standard logarithmic potential function

$$\Phi_t(\theta, f(x)) = \theta - \frac{t}{M} \sum_{m=1}^{M} \ln(\theta - f_m(x)),$$

where $\theta \in \mathbb{R}^+$, $f(x) = (f_1(x), \ldots, f_M(x))^T$ is the value of the function f for $x \in B$ and $t > 0$ is a tolerance (that depends on ϵ).

Let $\lambda_{\mathcal{P}}(f(x)) = \max\{f_1(x), \ldots, f_M(x)\}$. The potential function described above is used to calculate the price vector $p(f(x))$ for the block problem; $p(f(x))$ gives the direction in which we optimize in the next iteration.

For a fixed $f(x) \in \mathbb{R}^M_+$ and $x \in B$, the function $\Phi_t(\theta, f(x))$ has the barrier property: $\lim_{\theta \to \lambda_{\mathcal{P}}(f(x))+} \Phi_t(\theta, f(x)) = \infty$ and $\lim_{\theta \to \infty} \Phi_t(\theta, f(x)) = \infty$. Since $\Phi_t(\theta, f(x))$ is convex in θ, $\Phi_t(\theta, f(x))$ has an unique minimum $\theta(f(x))$ in $(\lambda_{\mathcal{P}}(f(x)), \infty)$. The minimum $\theta(f(x))$ can be determined by solving the equation

$$\frac{t}{M} \sum_{m=1}^{M} \frac{1}{\theta - f_m(x)} \ = \ 1. \tag{1}$$

In this case $g(\theta) = \frac{t}{M} \sum_{m=1}^{M} \frac{1}{\theta - f_m(x)}$ is a strictly decreasing function of θ in $(\lambda_{\mathcal{P}}(f(x)), \infty)$. Let $\phi_t(f(x)) = \Phi_t(\theta(f(x)), f(x))$ be the reduced potential value.

The dual vector (or price vector) $p(f(x))$ for a fixed vector $f(x)$ is defined by

$$p_m(f(x)) = \frac{t}{M} \frac{1}{\theta(f(x)) - f_m(x)} \qquad m = 1, \ldots, M. \tag{2}$$

In the following we describe an algorithm A_L to compute a solution for $(\mathcal{P}_\epsilon)$ and $(\mathcal{D}_\epsilon)$ in the case that we know a starting solution $x_0 \in B$ with $\lambda_{\mathcal{P}}(f(x_0)) \leq 1$. For an accuracy $t \in (0, 1)$, the algorithm is based on an approximate block solver (ABS) that solves the block problem approximately:

$$ABS(p,t) \qquad \text{compute } \hat{x} = x(p) \in B \text{ such that } p^T f(\hat{x}) \le (1+t)\Lambda(p)$$

where $p \in P$.

The algorithm A_L works for $\sigma \in (0, 1/3)$ as follows:

```
Algorithm $A_L(f, x_0, \sigma)$
(1) set $x := x_0$, $t := 3\sigma/7$, and $finished := false$,
(2) repeat
    (2.1) compute $f(x)$, $\theta(f(x))$ and $p(f(x))$;
    (2.2) set $\hat{x} := ABS(p(f(x)), \frac{t}{3})$;
    (2.3) if $p(f(x))^T f(\hat{x}) \ge \theta(f(x)) - 2t$ then set $finished := true$;
    (2.4) if $not(finished)$ then set $x := (1-\tau)x + \tau\hat{x}$, where $\tau = \frac{1}{M}\frac{4t^2}{25+10t}$
    until $finished = true$;
(3) return$(x, p(f(x)))$.
```

In the algorithm we have chosen as step length $\tau = \frac{1}{M}\frac{4t^2}{25+10t} \in (0,1]$. The step length is one of the critical parameters of the algorithm. It is chosen such that if the stopping criteria is not fulfilled then the reduced potential value $\phi_t(f(x'))$ for the next solution x' is smaller than the reduced potential value $\phi_t(f(x))$. One can prove the following inequality: $\phi_t(f(x)) - \phi_t(f(x')) \ge t^3/(13M)$. Since there is also an upper bound for the difference $\phi_t(f(x_0)) - \phi_t(f(x))$, we can show that the number of iterations is finite. In Section 6 we prove that algorithm A_L halts after

$$O(M(\sigma^{-3}(\lambda_{\mathcal{P}}(f(x_0)) - \lambda^*_{\mathcal{P}}) + \sigma^{-2}\ln\sigma^{-1}))$$

iterations for any initial $x_0 \in B$ with $\lambda_{\mathcal{P}}(f(x_0)) \le 1$ and any $\sigma \in (0, 1/3]$. Furthermore if the stopping criteria is fulfilled (i.e. $p(f(x))^T f(\hat{x}) \ge \theta(f(x)) - 2t$), then $\lambda_{\mathcal{P}}(f(x)) < \lambda^*_{\mathcal{P}} + \sigma$ and $\Lambda(p(f(x))) > \lambda^*_{\mathcal{P}} - \sigma$. This implies that x and $p(f(x))$ is an approximate solution of $(\mathcal{P})$ and $(\mathcal{D})$ with additive error σ, respectively.

Algorithm A_L can be used now also for the general case (without restrictions on the objective value $\lambda_{\mathcal{P}}(f(x_0))$). The algorithm uses a ternary search procedure TS and calls A_L as a subroutine several times. TS maintains an interval $[\underline{\lambda}, \overline{\lambda}]$ of length $\delta = \overline{\lambda} - \underline{\lambda}$ and two further parameters $\lambda_1 = \underline{\lambda} + \delta/3$ and $\lambda_2 = \underline{\lambda} + 2\delta/3$ that split the interval into three subintervals. Furthermore, let $v = 1/(2\underline{\lambda} + \overline{\lambda})$ and $\sigma = v\delta/3$.

Given an initial vector $x_0 \in B$ and accuracy $\epsilon > 0$, algorithm TS works as follows:

```
Algorithm $TS(x_0, \epsilon)$
(1) set $\underline{\lambda} = 0$, $x = x_0$, and $\overline{\lambda} = \lambda_{\mathcal{P}}(f(x_0))$,
(2) repeat
    (2.1) set $\delta = \overline{\lambda} - \underline{\lambda}$; $\lambda_1 = \underline{\lambda} + \delta/3$; $\lambda_2 = \lambda_1 + \delta/3$; $v = 1/(2\underline{\lambda} + \overline{\lambda})$; and
          $\sigma = v\delta/3$;
    (2.2) call algorithm $A_L(vf, x, v\delta/3)$ to get a point $y \in B$ such that
          $\lambda_{\mathcal{P}}(vf(y)) \le v\lambda^*_{\mathcal{P}} + \sigma$;
    (2.3) if $\lambda_{\mathcal{P}}(f(y)) \le \lambda_2$ then replace $x$ by $y$ and $\overline{\lambda}$ by $\lambda_2$ else replace $\underline{\lambda}$ by
          $\lambda_1$;
    until $\delta \le 3\epsilon\underline{\lambda}$;
(3) return($x$).
```

For a given function f and minimum value $\lambda^*_{\mathcal{P}} = \lambda^*_{\mathcal{P}}(f)$, algorithm A_L with function $\bar{f} = vf$ gives a point $y \in B$ such that $\lambda_{\mathcal{P}}(\bar{f}(y)) \le \lambda^*_{\mathcal{P}}(\bar{f}) + \sigma$. Now, $\lambda^*_{\mathcal{P}}(\bar{f}) = v\lambda^*_{\mathcal{P}}(f)$ and $\lambda_{\mathcal{P}}(\bar{f}(y)) = v\lambda_{\mathcal{P}}(f(y))$. Therefore, such a $y \in B$ satisfies $\lambda_{\mathcal{P}}(f(y)) \le \lambda^*_{\mathcal{P}}(f) + \sigma/v = \lambda^*_{\mathcal{P}}(f) + \delta/3$. Next notice that the calls to A_L are feasible. To show this, observe that $\sigma = v\delta/3 \le 1/3$ and $v\lambda_{\mathcal{P}}(f(x_0)) = \frac{1}{2\underline{\lambda}+\overline{\lambda}}\lambda_{\mathcal{P}}(f(x_0)) \le 1$. The next step is to show that TS generates a sequence of intervals $[\underline{\lambda}, \overline{\lambda}]$ where each interval contains $\lambda^*_{\mathcal{P}}$. To do this consider step (2.3) of the algorithm. If $\lambda_{\mathcal{P}}(f(y)) \le \lambda_2$, then $\lambda^*_{\mathcal{P}} \le \lambda_{\mathcal{P}}(f(y))$ and the replacement is feasible. If $\lambda_{\mathcal{P}}(f(y)) > \lambda_2$ then $\lambda^*_{\mathcal{P}} \ge \lambda_{\mathcal{P}}(f(y)) - \delta/3 > \lambda_2 - \delta/3 = \lambda_1$. In both cases after the replacement $\underline{\lambda} \le \lambda^*_{\mathcal{P}} \le \lambda_{\mathcal{P}}(f(x)) \le \overline{\lambda}$. Notice that we replace x by y only in the case $\lambda_{\mathcal{P}}(f(y)) \le \lambda_2$. Furthermore, the length of the interval is decreased by the multiplicative factor 2/3 in each iteration.

Grigoriadis and Khachiyan [15] proved that the ternary search halts in

$$O(M(\epsilon^{-2}\ln\epsilon^{-1} + \ln(\lambda_{\mathcal{P}}(f(x_0))/\lambda^*_{\mathcal{P}})))$$

iterations with an $x \in B$ such that $\lambda_{\mathcal{P}}(f(x)) \le (1+\epsilon)\lambda^*_{\mathcal{P}}$. An initial solution x_0 can be computed by one block optimization step $ABS(e', 1)$ where $e' = (1/M, \dots, 1/M)^T$. This solution x_0 satisfies the inequality $\lambda_{\mathcal{P}}(f(x_0)) \le 2M\lambda^*_{\mathcal{P}}$. Inserting this into the formula above shows that the algorithm TS halts in $O(M(\epsilon^{-2}\ln\epsilon^{-1} + \ln M))$ iterations. Finally, since $\theta(f(x))$ can not be in general computed exactly, we need a numerical overhead of $O(M\ln\ln(M\epsilon^{-1}))$ arithmetic operations per iteration to approximate the minima $\theta(f(x))$. For details we refer the reader to Section 6.

3 Max-Min Resource Sharing

In this section we present an algorithm for the max-min resource sharing problem proposed by Grigoriadis et al. [16]. Consider the following optimization problem $(\mathcal{C})$:

$$\begin{array}{lll} \text{Max } \lambda & & \\ \text{s.t. } f_i(x_1,\dots,x_N) \ \ge \ \lambda, & & i = 1,\dots,M, \\ \qquad (x_1,\dots,x_N) \in B, & & \end{array}$$

where $B \subset \mathbb{R}^N$ is a nonempty convex compact set and $f_i : B \to \mathbb{R}^+$, $i = 1,\dots,M$ are non-negative continuous concave functions on B. Let $f(x) = (f_1(x), \dots, f_M(x))^T$ for $x = (x_1,\dots,x_N)^T \in B$, and let

$$\lambda^*_{\mathcal{C}} = \max\{\lambda \mid f(x) \ge \lambda e, x \in B\}$$

where $e = (1,\dots,1)^T$. We are interested in computing an ϵ - approximate solution for $(\mathcal{C})$; i.e. for any error tolerance $\epsilon \in (0,1)$ we want to solve the problem

$$(\mathcal{C}_\epsilon) \qquad \text{compute } \ x \in B \ \text{ such that } \ f(x) \ \ge \ (1-\epsilon)\lambda^*_{\mathcal{C}} e.$$

In order to solve this max-min resource sharing problem we study the block problem

$$\Lambda(p) = \max \ \{ \ p^T f(x) \mid x \in B \ \}$$

for $p \in P = \{p \in \mathbb{R}^M \ : \ \sum_{i=1}^M p_i = 1, \ p_i \geq 0\}$. The block problem here is the maximization of a concave function over the convex set B. Suppose that we have an approximate block solver (ABS) that solves the block problem in the following way:

$$ABS(p,t) \qquad \text{compute } \ \hat{x} = \hat{x}(p) \in B \ \text{ such that } \ p^T f(\hat{x}) \ \geq \ (1-t)\Lambda(p).$$

The block solver $ABS(p,t)$ for $t > 0$ can be interpreted as a family of approximation algorithms for the block problem with ratio $1/(1-t)$. By duality we have

$$\lambda^*_{\mathcal{C}} = \max_{x \in B} \min_{p \in P} p^T f(x) = \min_{p \in P} \max_{x \in B} p^T f(x).$$

This implies that $\lambda^*_{\mathcal{C}} = \min\{\Lambda(p) \mid p \in P\}$. Based on this equality, one can naturally define the problem of finding an ϵ-approximate dual solution:

$$(\mathcal{D}_\epsilon) \qquad \text{compute } \ p \in P \ \text{ such that } \ \Lambda(p) \ \leq \ (1+\epsilon)\lambda^*_{\mathcal{C}}.$$

In this section we describe an approximation algorithm for the max-min resource sharing problem that computes a solution with objective value at least $(1-\epsilon)\lambda^*_{\mathcal{C}}$, provided that there is an approximate block solver $ABS(p,t)$ for $t = \Theta(\epsilon)$ and any $p \in P$. The algorithm that we describe in detail below solves $(\mathcal{C}_\epsilon)$, $(\mathcal{D}_\epsilon)$ in $O(M(\ln M + \epsilon^-2))$ iterations, where each iteration requires a call to $ABS(p, \Theta(\epsilon))$ and a coordination overhead of $O(M \ln\ln(M/\epsilon))$ operations for numerical computations. The algorithm uses the logarithmic potential function

$$\Phi_t(\theta, f(x)) = \ln\theta + \frac{t}{M} \sum_{m=1}^{M} \ln(f_m(x) - \theta),$$

where $\theta \in \mathbb{R}$, $f(x) = (f_1(x), \ldots, f_M(x))^T$ is the function value for an $x \in B$, and $t > 0$ is a tolerance (that depends on ϵ). For $\theta \in (0, \lambda_{\mathcal{C}}(f(x)))$ where $\lambda_{\mathcal{C}}(f(x)) = \min\{f_1(x), \ldots, f_M(x)\}$, the function Φ_t is well defined. The maximizer $\theta(f(x))$ of function $\Phi_t(\theta, f(x))$ is given by the first order optimality condition

$$\frac{t\theta}{M} \sum_{m=1}^{M} \frac{1}{f_m(x) - \theta} \ = \ 1. \qquad (3)$$

This equality has a unique root since $g(\theta) = \frac{t\theta}{M} \sum_{m=1}^M \frac{1}{f_m(x)-\theta}$ is a strictly increasing function of θ. The price vector $p = p(f(x))$ for a fixed $f(x)$ is defined by

$$p_m(f(x)) = \frac{t}{M} \frac{\theta(f(x))}{f_m(x) - \theta(f(x))} \qquad m = 1, \ldots, M. \qquad (4)$$

We can show that $p(f(x)) = (p_1(f(x)), \ldots, p_M(f(x))) \in P$ and $\lambda_{\mathcal{C}}(f(x))$ approximates $\theta(f(x))$ (see Section 7). The vector $p(f(x))$ is used in the approximate block solver $ABS(p(f(x)), t)$ as the next direction for the optimization. If $f_m(x) >> \theta(f(x))$, then $p_m(f(x)) \approx 0$. On the other hand, if $f_m(x)$ is close to $\theta(f(x))$, then the price component $p_m(f(x))$ is close to 1.

Let $\phi_t(f(x)) = \Phi_t(\theta(f(x)), f(x))$ be the reduced potential value.

Now we define a parameter $v = v(x, \hat{x})$ by

$$v(x, \hat{x}) = \frac{p^T f(\hat{x}) - p^T f(x)}{p^T f(\hat{x}) + p^T f(x)}, \tag{5}$$

where $p = p(f(x)) \in P$ and $\hat{x} \in B$ is an approximate block solution produced by $ABS(p(f(x)), t)$. Notice that $v(x, \hat{x}) \leq 1$. In Section 7 we show that if $v(x, \hat{x}) \leq t$ for $t = \epsilon/6$ (the stopping criteria), then x solves the primal problem $(\mathcal{C}_\epsilon)$ and the price vector $p(f(x))$, as defined in (4), solves $(\mathcal{D}_\epsilon)$.

The main algorithm works as follows:

```
**Algorithm** Improve($f, B, \epsilon, x$)
(1) set $t := \epsilon/6$; $v := t + 1$;
(2) while $v > t$ do end
    (2.1) compute $\theta(f(x))$ and $p = p(f(x)) \in P$ as defined in (3) and (4);
    (2.2) set $\hat{x} := ABS(p(f(x)), t)$;
    (2.3) compute $v(x, \hat{x})$;
    (2.4) if $v(x, \hat{x}) > t$ then set $x = (1-\tau)x + \tau\hat{x}$, where $\tau = \frac{t\theta(f(x))v(x,\hat{x})}{2M(p^T f(\hat{x}) + p^T f(x))}$
    end;
(3) return($x, p(f(x))$)
```

The algorithm Improve is a direct implementation of the Lagrangian decomposition scheme. The algorithm starts with an initial vector $x = x^0 = \frac{1}{M}\sum_{m=1}^{M} \hat{x}^{(m)}$ where $\hat{x}^{(m)}$ is the solution of $ABS(e_m, 1/2)$ and e_m is the unit vector with all zero coordinates except its m.th component which is 1. Then, the algorithm moves from x to $(1-\tau)x + \tau\hat{x}$ where $\hat{x}$ is the solution of $ABS(p(f(x)), t)$ until the stopping criteria $v(x, \hat{x}) \leq t$ is fulfilled. In Section 7 we prove that this algorithm solves $(\mathcal{C}_\epsilon)$ and $(\mathcal{D}_\epsilon)$ within $O(M\epsilon^{-1}(\ln M + \epsilon^{-1}))$ iterations (calls to the block solver). In order to prove this result, we show that

(1) the reduced potential values are monotone increasing from one vector x to the next vector $x' = (1-\tau)x + \tau\hat{x}$ (i.e. $\phi_t(f(x)) < \phi_t(f(x')))$,

(2) there is an upper bound for the difference $\phi_t(f(y)) - \phi_t(f(x))$ for any two vectors $x, y \in B$ with $\lambda_{\mathcal{C}}(f(x)), \lambda_{\mathcal{C}}(f(y)) > 0$.

The step length τ is chosen as $\frac{t\theta(f(x))v(x,\hat{x})}{2M(p^T f(\hat{x}) + p^T f(x))}$ to prove property (1) for the reduced potential values.

The total number of iterations can be improved by the scaling method used in [16,30]. The idea is to reduce the parameter t in different phases to the desired accuracy. In the s.th scaling phase we set $\epsilon_s = \epsilon_{s-1}/2$ and $t_s = \epsilon_s/6$ and use the current approximate point x^{s-1} as the initial solution. For phase $s = 0$, we use the initial point $x^0 \in B$. For this point we have $p^T f(x^0) \geq \frac{1}{2M}\Lambda(p)$ for each $p \in P$. We set $\epsilon_0 = (1 - 1/(2M))$. The initial solution satisfies $f_m(x^0) \geq \frac{1}{2M}\lambda^*_{\mathcal{C}} = (1 - 1 + \frac{1}{2M})\lambda^*_{\mathcal{C}} = (1 - \epsilon_0)\lambda^*_{\mathcal{C}}$ for each $m = 1, \ldots, M$. We show that this scaling implementation computes solutions x and p for $(\mathcal{C}_\epsilon)$ and $(\mathcal{D}_\epsilon)$ within $O(M(\ln M + \epsilon^{-2}))$ iterations. Similar to the min-max problem the root $\theta(f(x))$ can not in general be computed exactly. Therefore, we need a numerical overhead of $O(M \ln\ln(M\epsilon^{-1}))$ arithmetic operations per iteration to compute $\theta(f(x))$ approximatively. For the details of the analysis we refer the reader to Section 7.

4 Scheduling Jobs on Unrelated Machines

In this section we consider now a classical scheduling problem. Let $J = \{J_1, \ldots, J_n\}$ be a set of n jobs and $M = \{M_1, \ldots, M_m\}$ be a set of m machines. Suppose that the execution time of job J_j on machine M_i is $p_{ij} \geq 0$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. The objective is to find a schedule such that each job is processed by exactly one machine and the makespan (the largest completion time) is minimized.

Consider an example with 6 jobs and 3 machines and execution times as given in Table 1. A feasible schedule of length 8 is illustrated in Figure 1.

In the general case the problem is strongly NP-hard, and there is no polynomialtime $(1+\epsilon)$-approximation algorithm for any $\epsilon < \frac{1}{2}$, unless P=NP [28]. On the other hand, there is a polynomial-time 2-approximation algorithm proposed by Lenstra, Shmoys and Tardos [28]. For a fixed number of machines, the situation

Table 1. An example with 6 jobs and 3 machines

job J_j	J_1	J_2	J_3	J_4	J_5	J_6
p_{1j}	3	8	4	6	8	3
p_{2j}	5	2	5	6	4	3
p_{3j}	8	2	8	5	8	3

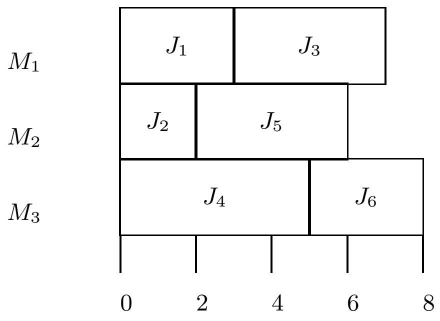

Fig. 1. A feasible schedule for the example

is better. Here the scheduling problem is weakly NP-hard (even for $m = 2$), and there is a fully polynomial time approximation algorithm (FPTAS) proposed by Horowitz and Sahni [18] that runs in $O(n^m(\frac{m}{\epsilon})^m)$ time. This running time has been improved by Jansen and Porkolab [19] to $O(n(\frac{m}{\epsilon})^{O(m)})$. Recently, Fishkin et al. [12] found a simpler and faster FPTAS that runs in $O(n + (\frac{\log m}{\epsilon})^{O(m^2)})$ time. One of the main open problems in scheduling is to close the gap between the lower bound (the inapproximability result) of 3/2 and the upper bound (the best known approximation ratio) of 2.

In the next subsection we describe in detail the 2 - approximation algorithm by Lenstra, Shmoys and Tardos [28]. The algorithm is based on linear programming. Afterwards, we show how to improve the running time of this algorithm using the min-max resource sharing framework [30,15].

4.1 A 2 Approximation Algorithm

The above scheduling problem can be described by the following integer linear program ILP(T):

$$\begin{array}{lll} \text{Min } \lambda & & \\ s.t. & \sum_{j=1}^{n} p_{ij} x_{ij} \le \lambda & i = 1, \dots, m \\ & \sum_{i=1}^{m} x_{ij} = 1 & j = 1, \dots, n \\ & x_{ij} \in \{0, 1\} & i = 1, \dots, m; j = 1, \dots, n \\ & x_{ij} = 0 & p_{ij} > T, \end{array}$$

where T is a guessed schedule length (see next paragraph). If a processing time $p_{ij} > T$ we set $x_{ij} = 0$. In any feasible solution of the ILP(T), $x_{ij} = 1$ indicates that job J_j is executed on machine M_i. The first m constraints are used to bound the total execution time on each machine by λ, and the next n constraints ensure that every job gets assigned to exactly one machine. The minimum makespan is given by the solution of ILP(T) where $T = \max_{i,j} p_{ij}$.

The algorithm in [28] is based on guessing the schedule length T, solving the linear program relaxation LP(T) of the above integer program and rounding the fractional variables $x_{ij} \in (0, 1)$ to zero or one. Given a deadline or target length T, the algorithm either deduces that there is no schedule with length at most T (i.a. when the objective value λ^* of the linear program is larger than T), or it constructs a schedule with makespan $2T$ (even if there is no schedule of length T). Notice that there are only nm different processing times p_{ij}. If T is fixed, variables x_{ij} with corresponding processing time $p_{ij} > T$ are set to zero (and eliminated). Therefore the number of choices for T can be restricted to the number of different processing times. Using binary search over these choices for T and the rounding procedure described below, we obtain an algorithm with approximation ratio 2. For the binary search we either sort the list of processing times in non-decreasing order and store subsequences of the ordered list or compute stepwise the median of a sublist of processing times. If the minimum makespan OPT is larger than $\max_{i,j} p_{ij}$ then the objective value of the LP relaxation for $T = \max_{i,j} p_{ij}$ gives a lower bound for OPT.

Suppose now that $OPT \leq \max_{i,j} p_{ij}$. The other case with $OPT > \max_{i,j} p_{ij}$ works in a similar way. In this case the rounding step described gives a schedule of length at most $OPT + T$. Given deadline $T \leq \max_{i,j} p_{ij}$, let $\mathcal{J}_i(T)$ be the set of jobs with execution time $p_{ij} \leq T$ on machine M_i (i.e. $\mathcal{J}_i(T) = \{J_j | p_{ij} \leq T\}$) and let $\mathcal{M}_j(T)$ be the set of machines that can process job J_j in time at most T (i.e. $\mathcal{M}_j(T) = \{M_i | p_{ij} \leq T\}$). Suppose that $\mathcal{M}_j(T) \neq \emptyset$; otherwise there is no schedule of length at most T. Consider the following system LP(T) of (in-)equalities:

$$\begin{array}{ll} \sum_{J_j \in \mathcal{J}_i(T)} p_{ij} x_{ij} \leq T & i = 1, \ldots, m \\ \sum_{M_i \in \mathcal{M}_j(T)} x_{ij} = 1 & j = 1, \ldots, n \\ x_{ij} \geq 0 & J_j \in \mathcal{J}_i(T), i = 1, \ldots, m \end{array}$$

Each zero - one assignment corresponding to a schedule with bounded execution times $p_{ij} \leq T$ is a feasible solution of this system of (in-)equalities. Therefore, if there is a schedule of length T, then this system is guaranteed to be feasible. A vertex of the polytope corresponding to the linear program $LP(T)$ has the property that the number of strict positive components is at most $n+m$ (the number of rows in $LP(T)$). This implies that each basic feasible solution has at most $n + m$ positive variables $x_{ij} > 0$; the other variables have value zero. Since each job J_j has at least one positive variable $x_{ij} > 0$, at most m jobs have been split onto two or more machines. In other words, $n - m$ jobs have an integral assignment $x_{ij} \in \{0, 1\}$ for $i = 1, \ldots, m$. These jobs are assigned already to an unique machine.

Given a basic feasible solution (x^*_{ij}) of LP(T), we construct now a bipartite graph $G(x^*) = (V, E)$ with vertex set $V = J \cup M$ (where $J = \{J_1, \ldots, J_n\}$ and $M = \{M_1, \ldots, M_m\}$) and edge set $E = \{(J_j, M_i) | x^*_{ij} > 0\}$. By the arguments above, $|E| \leq |V|$. This holds also for each connected component of G as shown below.

Lemma 1. *Let C be a connected component of $G(x^*)$. The number of edges in C is bounded by the number of vertices in C.*

Proof. Let M_C and J_C be the set of machine and job nodes contained in C, respectively. Furthermore, let x^*_C denote the restriction of x^* to variables x^*_{ij} with $M_i \in M_C$ and $J_j \in J_C$, and let $x^*_{\bar{C}}$ denote the remaining variables of x^*. Reorder the variables such that $x^* = (x^*_C, x^*_{\bar{C}})$. Consider now the following restricted system of (in-)equalities:

$$\sum_{J_j \in \mathcal{J}_i(T) \cap J_C} p_{ij} x_{ij} \leq T \qquad M_i \in M_C$$

$$\sum_{M_i \in \mathcal{M}_j(T) \cap M_C} x_{ij} = 1 \qquad J_j \in J_C$$

$$x_{ij} \geq 0 \qquad J_j \in J_C, M_i \in M_C, \text{where } p_{ij} \leq T.$$

We prove now that x^*_C is a vertex of the polytope corresponding to this system of linear (in-)equalities. Suppose that this is not the case. Then there exist y_1, y_2 with $x^*_C = (y_1 + y_2)/2$ where y_i are feasible solutions of the polytope. But then, $x^* = ((y_1/2, x^*_{\bar{C}}/2) + (y_2/2, x^*_{\bar{C}}/2))$ where $(y_i, x^*_{\bar{C}})$ are feasible solutions of the original polytope. Since x^* is a vertex, this gives a contradiction. Therefore, x^*_C is also a vertex of the restricted polytope and the connected component C has no more edges than vertices.

Since $C = G(x^*_C)$ is connected, C is either a tree or a tree plus an additional edge. The latter graph is called a 1-tree (see also Figure 2) . This implies that $G = G(x^*)$ is a pseudo forest (i.e. a forest of trees and 1-trees). For the rounding we delete first all jobs with integral assignments. This generates a pseudo forest where all leaves correspond to machine nodes. Each remaining fractional job

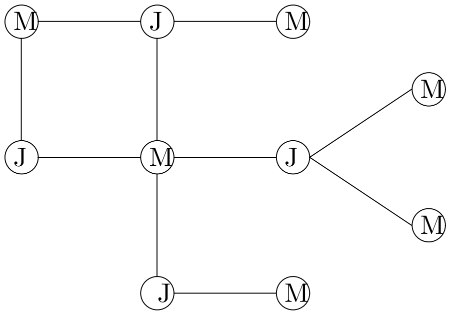

Fig. 2. One of the 1-trees in a pseudo forest

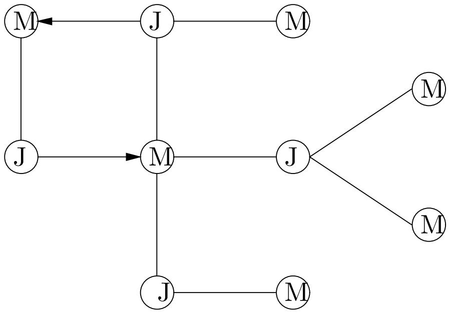

Fig. 3. The orientation along the cycle in the 1-tree

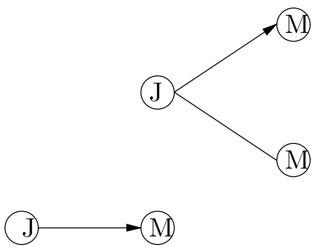

Fig. 4. The remaining trees and assignment of job to machine nodes

node has at least two incident edges. The next step is to eliminate the cycles in the remaining graph G' and to construct a matching in G' in which every job node gets matched to an unique machine. Using such a matching, each machine will have load at most $2T$ (i.e. a load of at most T by the integral assignment and at most 1 further job by the matching).

We construct the matching in the following way: for each remaining 1-tree we determine the unique cycle in it and orient the cycle in one direction (see also Figure 3). Using this orientation, each job node in the cycle is assigned to its succeeding machine node. After that we delete all nodes along the cycles. In this way we obtain a set of trees with at most one job leaf per resulting tree. Here we use the fact that the original 1-trees have only leaves corresponding to machine nodes. Finally we root each tree at its unique job leaf or at an arbitrary job node (if no job leaf exists) and assign each job node to one of its children (that is always a machine node). An example is given in Figure 4. This gives an assignment where each machine gets at most one job and each job is matched to exactly one machine. Since each processing time $p_{ij} \le T$ for $M_i \in M_j(T)$, the schedule corresponding to this assignment has length at most T.

4.2 A Faster $(2+\epsilon)$ Approximation Algorithm

The main bottleneck in the approximation algorithm is the time needed to solve the linear program LP(T). Let us consider the min-max resource sharing approach with coupling constraints:

$$\begin{array}{ll} \text{Min} & \lambda \\ f_i(x) = \sum_{j=1}^{n} p_{ij} x_{ij} \le \lambda, & \qquad i = 1, \ldots, m, \end{array}$$

and block $B = B^1 \times \ldots \times B^n$ where B^j is a simplex that contains the variables $(x_{ij})_{i=1,\ldots,m}$ and that is given by the following (in-)equalities:

$$\sum_{i=1}^{m} x_{ij} = 1,$$

$$x_{ij} = 0 \quad for\, p_{ij} > T$$

$$x_{ij} \ge 0 \quad for\, p_{ij} \le T.$$

In the block optimization we have to compute for a given price vector $y = (y_1, \ldots, y_m) \in P$:

$$\Lambda(y) = \min\{y^T f(x) | x \in B\}.$$

Since $B = B^1 \times \ldots \times B^n$, the minimization can be done independently (or separately) over each B^j, $j = 1, \ldots, n$. For each simplex B^j we have to compute the minimum modified processing time $y_i p_{ij}$ over machines i with $p_{ij} \leq T$. Each block optimization takes $O(mn)$ time. Furthermore, the number of iterations to find an assignment $\bar{x}$ with length $\leq (1+\epsilon)T$ is $O(m(\ln m + \epsilon^{-2} \ln \epsilon^{-1}))$, if one assignment of length T exists. The overhead (the numerical calculations to compute the root $\theta(f(x))$ approximately) in each iteration is at most $O(m \ln\ln(m\epsilon^{-1}))$. This gives a running time of at most

$$O(m^2(\ln m + \epsilon^{-2} \ln \epsilon^{-1}) \max(n, \ln\ln(m\epsilon^{-1})))$$

to approximately solve the linear program. Using the recent result by Jansen and Zhang [22], the running time can be slightly improved to

$$O(m^2(\ln m + \epsilon^{-2}) \max(n, \ln\ln(m\epsilon^{-1}))).$$

Using other variants of algorithms for the min-max resource sharing problem, we can achieve other running times (with dependence on n and m). For example, the number of iterations in the exponential-potential method with K disjoint blocks and separable coupling constraints is $O(K \ln M(\epsilon^{-2} + \ln M))$ [15]. This can be used directly for our scheduling problem (where $K = n$ and $M = m$). The number of iterations is $O(n \ln m(\epsilon^{-2} + \ln m))$. This would give a running time of

$$O(nm \ln m(\epsilon^{-2} + \ln m) \max(n, \ln\ln(m\epsilon^{-1}))).$$

This was further improved by Villavicencio and Grigoriadis [35] by simplification of the block optimization. In their approach, it is sufficient to optimize over only one block instead over K blocks.

However, in order to apply the rounding procedure, the approximate solution $\bar{x} = (\bar{x}_{ij})$ of the linear program must be converted into a modified solution $\hat{x} = (\hat{x}_{ij})$. For the solution $\bar{x}$ of $LP(T)$ we construct again a bipartite graph $G(\bar{x})$ as before.

Lemma 2. *Let $\bar{x}$ be a solution of the linear program $LP(T)$ represented by a bipartite graph $G(\bar{x}) = (V, E)$ with $V = J \cup M$. Then $\bar{x}$ can be converted in $O(|E|m) = O(nm^2)$ time into another solution $\hat{x}$ of the same length where the corresponding bipartite graph $G(\hat{x})$ is a forest.*

Proof. The main idea is to eliminate cycles in the bipartite graph. First consider the case where G contains of only one cycle $e_1, \ldots, e_{2r}$ (see also Figure 5). The idea is to perturb the values along the cycle to force at least one coordinate of the assignment $\bar{x}$ to be 0. We can always either increase the coordinate of the assignment $\bar{x}$ for each edge e_{2i} and decrease those for each edge e_{2i-1}, or vice versa. If e_i and e_{i+1} meet at a job node the perturbation must have the same

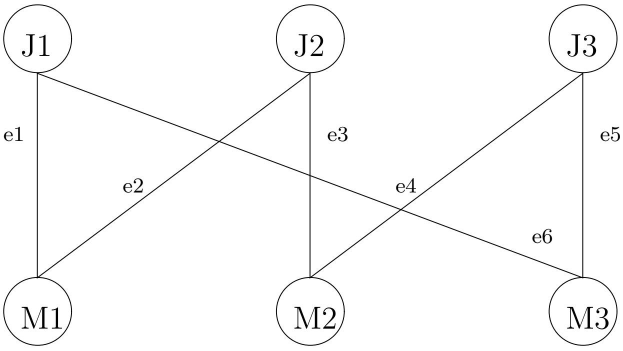

Fig. 5. Assignment of job nodes to machine nodes

magnitude. If e_i and e_{i+1} meet at a machine node, the perturbation is linearly related to the load constraint.

Consider the cycle in Figure 5. Suppose that we add $\delta_1 > 0$ on edge $e_1 = (J_1, M_1)$, $\delta_2 > 0$ on edge $e_3 = (J_2, M_2)$ and $\delta_3 > 0$ on edge $e_5 = (J_3, M_3)$. In this case we have to decrease the edge $e_6 = (J_1, M_3)$ by δ_1, edge $e_2 = (J_2, M_1)$ by δ_2 and edge $e_4 = (J_3, M_2)$ by δ_3. In other words, we add along the cycle $(e_1, \ldots, e_6)$ the following values $(\delta_1, -\delta_2, \delta_2, -\delta_3, \delta_3, -\delta_1)$. In order to satisfy the load constraints, the following inequalities must be satisfied:

$$\delta_1 p_{11} - \delta_2 p_{12} \leq 0$$

$$\delta_2 p_{22} - \delta_3 p_{23} \leq 0$$

$$\delta_3 p_{33} - \delta_1 p_{31} \leq 0.$$

These inequalities imply that

$$\delta_2 \geq \delta_1 \tfrac{p_{11}}{p_{12}}$$

$$\delta_3 \geq \delta_2 \tfrac{p_{22}}{p_{23}}$$

$$\delta_1 \geq \delta_3 \tfrac{p_{33}}{p_{31}}.$$

Such a choice of values for δ_i is possible, if

$$\delta_1 \geq \frac{p_{33}}{p_{31}} \frac{p_{22}}{p_{23}} \frac{p_{11}}{p_{12}} \delta_1,$$

or equivalent if $\frac{p_{33}}{p_{31}} \frac{p_{22}}{p_{23}} \frac{p_{11}}{p_{12}} \leq 1$. Furthermore, if this condition is satisfied, we can always define values for δ_i that satisfy the load constraints (e.g. $\delta_2 = \delta_1 \frac{p_{11}}{p_{12}}$, $\delta_3 = \delta_2 \frac{p_{22}}{p_{23}}$ for a given δ_1). In this way all constraints are satisfied. Now we compute a value for δ_1 that forces at least one coordinate of $\bar{x}$ to be 0. Given the approximate LP-solution $\bar{x} = (\bar{x}_{ij})$ we define

$$\delta_1 = \min\{\bar{x}_{31}, \bar{x}_{12} \frac{p_{12}}{p_{11}}, \bar{x}_{23} \frac{p_{12}}{p_{11}} \frac{p_{23}}{p_{22}}\}.$$

By adding the values along the cycle we get

$$x_{11} = \bar{x}_{11} + \delta_1$$

$$x_{12} = \bar{x}_{12} - \delta_1 \tfrac{p_{11}}{p_{12}}$$

$$x_{22} = \bar{x}_{22} + \delta_1 \tfrac{p_{11}}{p_{12}}$$

$$x_{23} = \bar{x}_{23} - \delta_1 \tfrac{p_{11}}{p_{12}} \tfrac{p_{22}}{p_{23}}$$

$$x_{33} = \bar{x}_{33} + \delta_1 \tfrac{p_{11}}{p_{12}} \tfrac{p_{22}}{p_{23}}$$

$$x_{31} = \bar{x}_{31} - \delta_1.$$

It is not complicated to check that all variables $x_{ij} \geq 0$ and $x_{ij} \leq \bar{x}_{ij} + \bar{x}_{i'j}$ where $M_{i'}$ is the other machine corresponding to job J_j in the cycle. For example, using $\delta_1 \leq \bar{x}_{31}$ we have $x_{31} \geq 0$ and $x_{11} \leq \bar{x}_{11} + \bar{x}_{31}$. In other words, the new solution is also a feasible solution of $LP(T)$. We have $x_{ij} \geq 0$, $\sum_{i=1}^{M} x_{ij} = 1$ and $\sum_{j=1}^{n} p_{ij} x_{ij} \leq T$. For example consider the constraints corresponding to job J_1 and machine M_1. We have $x_{11} + x_{31} = \bar{x}_{11} + \delta_1 + \bar{x}_{31} - \delta_1 = \bar{x}_{11} + \bar{x}_{31}$ and $p_{11}x_{11} + p_{12}x_{12} = p_{11}\bar{x}_{11} + p_{11}\delta_1 + p_{12}\bar{x}_{12} - p_{12}\delta_1 \frac{p_{11}}{p_{12}} = p_{11}\bar{x}_{11} + p_{12}\bar{x}_{12}$. This implies that the same total fraction of job J_1 is assigned to machines M_1 and M_3 and that the load on machine M_1 has not been changed. The same holds for the other jobs and machines. Furthermore, using the choice of δ_1 at least one of variables x_{12}, x_{23} or x_{31} will be zero.

On the other hand, if $\frac{p_{33}}{p_{31}} \frac{p_{22}}{p_{23}} \frac{p_{11}}{p_{12}} \geq 1$, then we can increase the coordinates of $\bar{x}$ for each edge e_2, e_4, e_6 and decrease the others. This means that we will add $(-\delta_1, \delta_2, -\delta_3, \delta_3, \delta_1)$ with $\delta_i \geq 0$ along the cycle $(e_1, \ldots, e_6)$. In this case we compute

$$\delta_1 = \min\{\bar{x}_{11}, \bar{x}_{12} \frac{p_{12}}{p_{11}}, \bar{x}_{22} \frac{p_{22}}{p_{23}} \frac{p_{33}}{p_{31}}\},$$

and set $\delta_3 = \delta_1 \frac{p_{31}}{p_{33}}$ and $\delta_2 = \delta_3 \frac{p_{23}}{p_{22}}$. Similar to the other case, we can compute a feasible solution x_{ij} of $LP(T)$ and can force at least one coordinate of the cycle to be 0.

By a modified depth first search we can generalize the above procedure to generate a solution $\hat{x}$ where the bipartite graph $G(\hat{x})$ is a forest. When a cycle is detected by a back edge, the perturbation is computed for the cycle and the search is restarted. When we detect that an edge does not belong to any cycle in G, then we store the information of the edge (i.e. the assignment for the corresponding job) and delete the edge to avoid repeatedly searching through this part of the graph. Since the depth of the search tree is at most $2m$, after $O(m)$ steps we either find a cycle or delete an edge that is not in any cycle. In both cases we delete at least one edge. Therefore, the total search time is $O(|E|m)$. Furthermore, the time to compute a perturbation along a cycle can be bounded by $O(m)$. This implies that the total running time of the rounding procedure is $O(|E|m) = O(nm^2)$.

5 Strip Packing

In this section we consider the problem to pack rectangles into a strip (strip packing). A rectangle r_i is given by its width $w(r_i) = w_i$ and height $h(r_i) = h_i$ where $0 \leq w_i, h_i \leq 1$. Let L be a list of n rectangles $r_i = (w_i, h_i)$. Given a strip of width 1, the strip packing problem is to find a packing of the rectangles $r_i \in L$ into the strip with minimum height such that all rectangles have disjoint interiors. The height of a strip packing is the uppermost boundary among all rectangles in the packing. Here rotations of rectangles are not allowed. We suppose that the list L of rectangles is ordered by non-increasing widths: $w_1 \geq w_2 \geq \ldots \geq w_n$.

A related problem is a scheduling problem with parallel multiprocessor tasks. Suppose there is given a set of n tasks $T = \{T_1, \ldots, T_n\}$ and a set of m identical processors $M = \{M_1, \ldots, M_m\}$. Each task T_j has to be processed on $size_j$ processors (where $size_j \in \{1, \ldots, m\}$) with execution time p_j. Given $size_j$ processors allotted to task T_j, these processors are required to execute task T_j in parallel and without preemption, i.e. they all have to start processing task T_j at some starting time τ_j and to complete it at $\tau_j + p_j$. Each processor can execute at most one job per time. The objective is to find a feasible non-preemptive schedule that minimizes the overall makespan; i.e. the maximum completion time among all tasks:

$$\max\{\tau_j + p_j | j = 1, \ldots, n\}.$$

This problem is called non-malleable parallel task scheduling (NPTS). An example with 7 tasks and $size_1 = \ldots = size_4 = size_7 = 1$, and $size_5 = size_6 = 2$ on three processors with processing times $p_1 = 5$, $p_2 = 2.5$, $p_3 = 1$, $p_4 = 1.5$, $p_5 = 4$, $p_6 = 4$ and $p_7 = 2$ is given in Figure 6. Notice that in general tasks can be executed on any set of processors with cardinality $size_j$ (i.e. the processor set do not need to be consecutive). The length or makespan of the schedule in Figure 6 is 10.5. NPTS is strongly NP-hard even for any constant number $m \geq 5$ of processors [11]. Furthermore, 2 is the best known approximation ratio for the problem [13]. If the number m of processors is fixed, then there is a polynomial time approximation scheme (PTAS) [20,2].

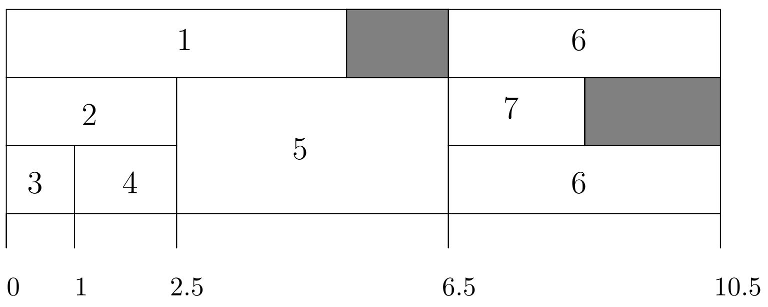

Fig. 6. A schedule of parallel tasks

Notice that the strip packing problem is equivalent to NPTS with consecutive processor allocation for tasks. Furthermore, there are other versions of NPTS in which the processors are arranged according to a given network (e.g. a hypercube or a mesh) [8,29,37] and tasks must be executed on a certain subset (e.g. on a subcube or a submesh). An example is illustrated in Figure 7 where each submesh corresponds to a rectangle of processors. In this case the scheduling problem is equivalent to a 3-dimensional packing problem. For a survey on scheduling parallel multiprocessor tasks we refer the reader to [10].

For a particular input list L of rectangles, let $OPT(L)$ be the minimum height, and let $A(L)$ denote the packing height obtained by algorithm A. The absolute performance ratio of A is $sup_L\, A(L)/OPT(L)$. The asymptotic performance ratio of A is

$$limsup_{OPT(L)\to\infty}\, A(L)/OPT(L).$$

An asymptotical fully polynomial approximation scheme is a family of approximation algorithms A_ϵ with asymptotic performance ratio $1+\epsilon$ and running time polynomial in the length of L and $\frac{1}{\epsilon}$. In the following we describe in details an asymptotical fully polynomial time approximation scheme (AFPTAS) for the strip packing problem with bounded rectangle heights $h_j \leq 1$ [25]. The algorithm computes a packing of height at most $(1+\epsilon)OPT(L)+O(1/\epsilon^2)$. Notice that the algorithm works also for NPTS with or without consecutive processor allocation for tasks. For other results on strip packing we refer the reader to [3,7,31,32].

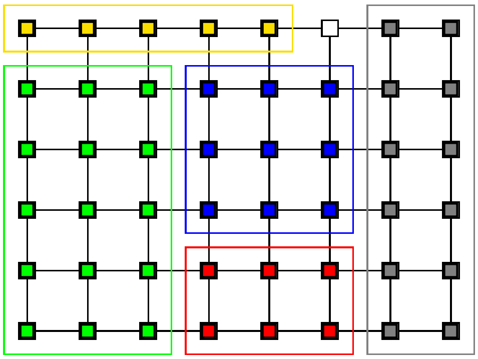

Fig. 7. 2-dimensional mesh of processors

Let h_{max} denote the maximum height of any rectangle in L. We suppose that the maximum height $h_{max} \leq 1$. Next we describe the main ingredients of the AFPTAS, and later we show how the max-min resource sharing approach helps to improve the running time of this approximation scheme.

The approximation scheme works as follows:

(1) it partitions the list $L=(r_1,\ldots,r_n)$ of rectangles into a set of narrow rectangles $L_{narrow}=\{r_i|w(r_i)\leq\epsilon'\}$ and a set of wide rectangles $L_{wide}=\{r_i|w(r_i)>\epsilon'\}$ where $\epsilon'=\epsilon/(2+\epsilon)$,

(2) it rounds the set L_{wide} into a similar instance L_{sup} that has only $O(1/\epsilon^2)$ different widths,
(3) it solves approximately the fractional strip packing problem (see next subsection) for L_{sup} and constructs a feasible strip packing for L_{sup},
(4) it packs the rectangles in L_{narrow} using a greedy algorithm into the remaining space.

5.1 Fractional Strip Packing

A fractional strip packing of L is a packing of any list L' obtained from L by replacing some of its rectangles $r_i = (w_i, h_i)$ by a sequence of rectangles (w_i, h_{i_1}), (w_i, h_{i_2}), ..., (w_i, h_{i_k}) where $h_{i_1}, \ldots, h_{i_k} \geq 0$ and $\sum_{\ell=1}^{k} h_{i_\ell} = h_i$. Some of the ideas below are based on a rounding technique for bin packing [9] and on other approximation algorithms for rectangle packings [3,7]. In step (3) of the algorithm we suppose that all n rectangles have only M different widths $w'_1 > w'_2 > \ldots > w'_M > \epsilon'$. Similar to bin packing we associate a set of configurations (multisets) that could occur in a fractional strip packing. A configuration is a multiset $\{\alpha_{1j} : w'_1, \alpha_{2j} : w'_2, \ldots, \alpha_{Mj} : w'_M\}$ where α_{ij} denotes the number of occurrences of width w'_i in configuration C_j such that $\sum_{i=1}^{M} \alpha_{ij} w'_i \leq 1$. Let q be the number of different configurations.

For each fractional strip packing P of total height h, we can compute a vector $(x_1, \ldots, x_q)$ with $x_i \geq 0$ associated to P. Scan the packing bottom up with a horizontal sweep line $y = a$ where $0 \leq a \leq h$. Each line corresponds to one of the configurations. Therefore, each fractional strip packing can be described by a sequence of configurations $C_{i_1}, \ldots, C_{i_k}$ with heights $h_{i_1}, \ldots, h_{i_k}$ such that $i_\ell \in \{1, \ldots, q\}$ for $\ell \in \{1, \ldots, k\}$ and $\sum_{\ell=1}^{k} h_{i_\ell} = h$. For each configuration C_j let $x_j = \sum_{\ell | C_j = C_{i_\ell}} h_{i_\ell}$ be the total height of C_j in P.

Table 2. The different configurations with rectangles of type A and B

	configuration	α_{1j} = number of A's	α_{2j} = number of B's
C_1	$\frac{3}{7}, \frac{2}{7}, \frac{2}{7}$	1	2
C_2	$\frac{3}{7}, \frac{3}{7}$	2	0
C_3	$\frac{2}{7}, \frac{2}{7}, \frac{2}{7}$	0	3
C_4	$\frac{3}{7}, \frac{2}{7}$	1	1
C_5	$\frac{2}{7}, \frac{2}{7}$	0	2
C_6	$\frac{3}{7}$	1	0
C_7	$\frac{2}{7}$	0	1

In Figure 8 we have a packing P with rectangles of type A with width 3/7 and height 1 and rectangles of type B with width 2/7 and height 3/4. The possible configurations with these types are given in Table 2. The vector corresponding to the strip packing is: $(3/2, 5/4, 0, 0, 0, 0, 0)$.

In the next part we study the fractional strip packing problem. Let β_i be the total height of all rectangles in the instance with width w'_i. The fractional strip

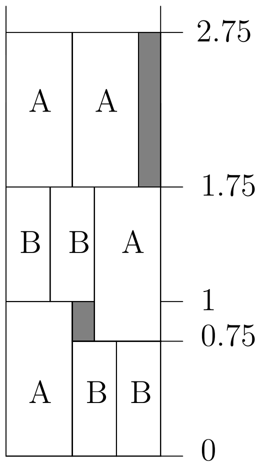

Fig. 8. A packing with 4 rectangles of type A and B

packing problem can be solved via linear programming. Use for each configuration C_j a variable $x_j \geq 0$ that denotes the total height of C_j in the fractional strip problem. Then, LP(L) is given by

$$\begin{array}{lll} \text{Min } \sum_{j=1}^{q} x_j & & \\ \text{s.t. } \sum_j \alpha_{ij} x_j \geq \beta_i & & i = 1, \ldots, M \\ x_j \geq 0 & & j = 1, \ldots, q. \end{array}$$

We denote with $LIN(L)$ the minimum objective value of LP(L) for a given list L. Fractional strip packing is closely related to fractional bin packing where the vector β in $LP(L)$ is a vector $(n_1, \ldots, n_M)$ with integer components. In bin packing we have a list of items of sizes $a_1 > \ldots > a_M > \epsilon'$ with n_i items of size a_i, for $i = 1, \ldots, M$. A configuration C_j here is a multiset $\{\alpha_{1j} : a_1, \ldots, \alpha_{Mj} : a_M\}$ where α_{ij} denotes the number of items of size a_i in C_j. The fractional bin packing problem can be solved approximately with additive tolerance t (i.e. obtaining an objective value $LIN(L) + t$) in time

$$O(M^6 \ln^2 \frac{Mn}{at} + \frac{M^5 n}{t} \ln \frac{Mn}{at})$$

using the algorithm by Karmarkar and Karp [24,26] (where n is the number of items, M is the number of distinct sizes, and a is the size of the smallest item). The algorithm is based on linear programming techniques (ellipsoid algorithm and separation oracle), but does not use the fact that the vector $(n_1, \ldots, n_M)$ is integer. Therefore, it can be used for fractional strip packing, too. This means that the fractional strip packing problem can be solved approximately with additive tolerance $t = 1$ in time polynomial in M, n and $1/\epsilon$. The number of non-zero

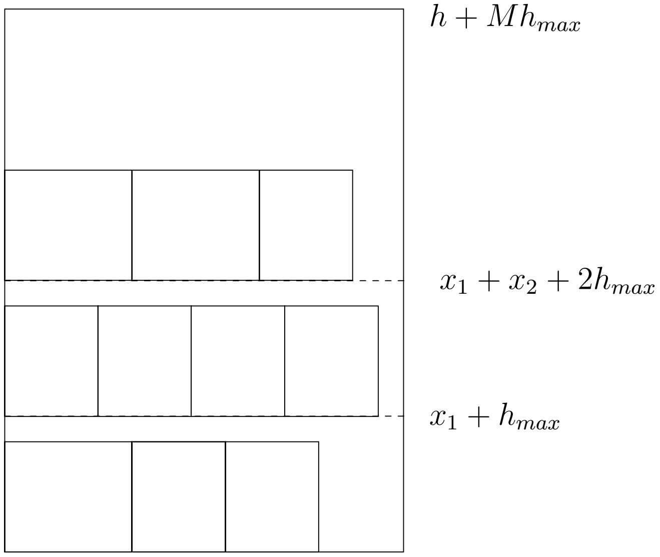

Fig. 9. Reserving space for wide rectangles

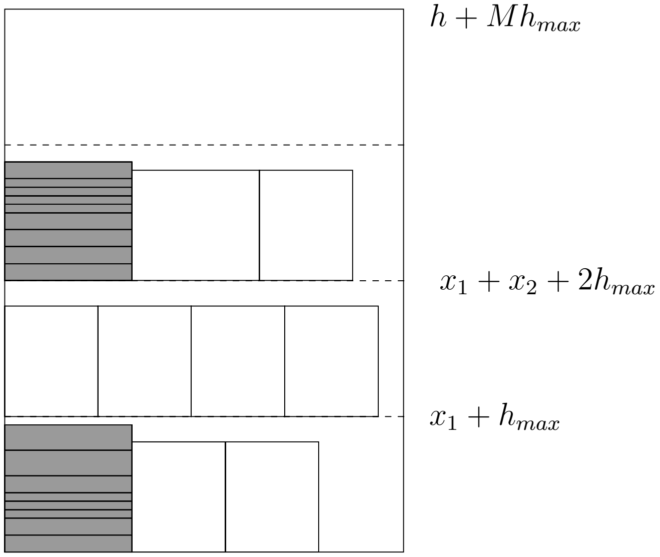

Fig. 10. Packing wide rectangles into the reserved space

variables (that is the number of configurations used) in the solution of LP(L) is at most M. In the next Lemma we show how to transform a general fractional solution into an integral solution.

Lemma 3. *Let $x = (x_1, \ldots, x_q)$ be a solution of LP(L) with objective value h and at most M non-zero variables x_j. Then we can transform such a solution into an integral solution or packing with height at most $h + Mh_{max} \leq h + M$.*

Proof. Without loss of generality we assume that $x_1, \ldots, x_M > 0$ and $x_{M+1} = \ldots = x_q = 0$. We construct an integral packing of height $h + Mh_{max} \leq h + M$ as follows: First cut the strip into M regions $Q_1, \ldots, Q_M$ of heights $x_j + h_{max}$ between levels $\ell_j = \sum_{k=1}^{j-1} x_k + (j-1)h_{max}$ and $\ell_{j+1} = \sum_{k=1}^{j} x_k + jh_{max}$ (see also Figure 9). The region Q_j corresponds to configuration C_j. Each region Q_j is divided into α_{ij} columns of height $x_j + h_{max}$ and width w'_i, for each $i = 1, \ldots, M$.

Then we place rectangles of width w'_i by a greedy algorithm into columns of the same width until the height of a column is larger than x_j or the list of rectangles is exhausted (see also Figure 10). We can pack all rectangles into the reserved space, since $\sum_j \alpha_{ij} x_j$ is at least the total height β_i of all rectangles of width w'_i. Suppose that not all rectangles fit into the columns. This implies that all columns of width w'_i are filled up to a value larger than x_j. Counting over all columns of width w'_i we obtain a total height larger than $\sum_j \alpha_{ij} x_j$. Since this sum is at least β_i, we get a contradiction.

The next goal is to obtain a nice layout of the integral packing. Let $c_1 \geq c_2 \geq \ldots \geq c_{M'}$ denote the total widths of the wide rectangles configurations C_j, for $j = 1, \ldots, M'$. If each layer $[0, 1] \times [\ell_j, \ell_{j+1}]$ can be divided into three regions R_j, R'_j, R''_j such that

(i) $R_j = [c_j, 1] \times [\ell_j, \ell_{j+1}]$ is completely free and will be used later to place narrow rectangles,
(ii) $R'_j = [0, c_j] \times [\ell_j, \ell_{j+1} - 2]$ is completely filled by wide rectangles, and
(iii) $R''_j = [0, c_j] \times [\ell_{j+1} - 2, \ell_{j+1}]$ is partially filled by wide rectangles and those free space is not used later for narrow rectangles,

then we call the integral packing nice. Condition (ii) is important for the analysis of the algorithm.

Lemma 4. *Let $x = (x_1, \ldots, x_q)$ be a solution of LP(L) with objective value h and at most M non-zero variables x_j. Then we can transform such a solution into a nice packing of height at most $h + M'h_{max} \leq h + M'$ and with at most $M' = 2M$ configurations.*

Proof. Let $x = (x_1, \ldots, x_q)$ be the vector as above. Now we cut the strip into M regions $Q_1, \ldots, Q_M$ of heights $x_1, \ldots, x_M$. As above, each region Q_j (corresponding to configuration C_j) is divided into α_{ij} columns of widths w'_i, for all $i = 1, \ldots, M$. Then we place the rectangles of width w'_i into the columns until exactly height x_j. In this way, each column contains a sequence of rectangles that completely fit inside the column and possibly the top part of a rectangle which had been started in a previous column and the bottom part of a rectangle which is too tall to fit into this column.

If we have too much space in the columns of width w'_i, then we distribute the rectangles among the columns and erase the extra part. We split the configuration C_j into two configurations: one configuration of the old type where the

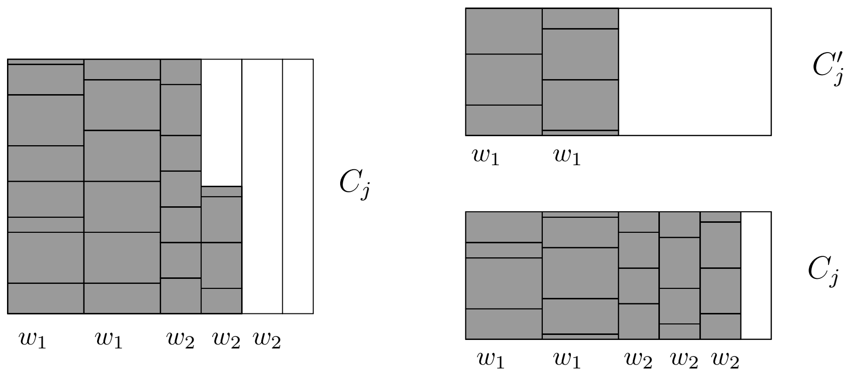

Fig. 11. Splitting a configuration C_j

columns of width w'_i are completely filled and one configuration without columns of width w'_i (see Figure 11). Such a case can happen at most M times. Therefore, the total number M' of configurations after this transformation is $2M$. The consequence is that all columns are completely used by rectangles. After that we reserve space for the M' configurations with regions $Q_1, \ldots, Q_{M'}$ between levels ℓ_j and ℓ_{j+1} of height $x_j + h_{max}$.

Each column C of the fractional strip packing of width w'_i and height x_j is associated to a column C_+ of width w'_i and height $x_j + h_{max}$. In C_+ we place all rectangles which fit completely in C (by the previous step) and the rectangle whose bottom is in C and whose top is another column. By the above construction, there is at most one rectangle of this type and the total height of rectangles placed in C_+ is at least $x_j - h_{max} \geq x_j - 1$. Therefore, each column C_+ is completely filled by wide rectangles until $\ell_j + x_j - 1 \geq \ell_{j+1} - 2$. This implies that region R'_j is completely filled by wide rectangles. Furthermore, region R_j is completely free and R''_j is partially filled by wide rectangles. Therefore, we have a nice integral packing.

Now we describe the rounding technique for the general case. Remember that $L_{narrow} = \{r_i | w_i \leq \epsilon'\}$ and $L_{wide} = \{r_i | w_i > \epsilon'\}$. The next goal is to round the rectangles in L_{wide} to obtain an instance L_{sup} with a constant number of different widths, where the optimum objective function values for the fractional strip packing instances satisfy $LIN(L_{sup}) \leq (1+\epsilon)LIN(L_{wide})$.

Definition 1. *Given two sequences $L = \{r_1, \ldots, r_n\}$ and $L' = \{r'_1, \ldots, r'_n\}$ of rectangles. We say that $L \leq L'$ if there is a bijection $f : L \to L'$ such that the width $w(r_i) \leq w(f(r_i))$ and height $h(r_i) \leq h(f(r_i))$ for $i = 1, \ldots, n$.*

The main idea now is to approximate L_{wide} by a list L_{sup} such that $L_{wide} \leq L_{sup}$ and that L_{sup} has only M different rectangle widths. To construct L_{sup} we build a stack packing as follows: we stack up all rectangles of L_{wide} by order of non-increasing widths on a left-justified stack of height $H = h(L_{wide})$ (see also Figure 12). We suppose that $h(L_{wide})$ is larger than a constant $O(M) =$

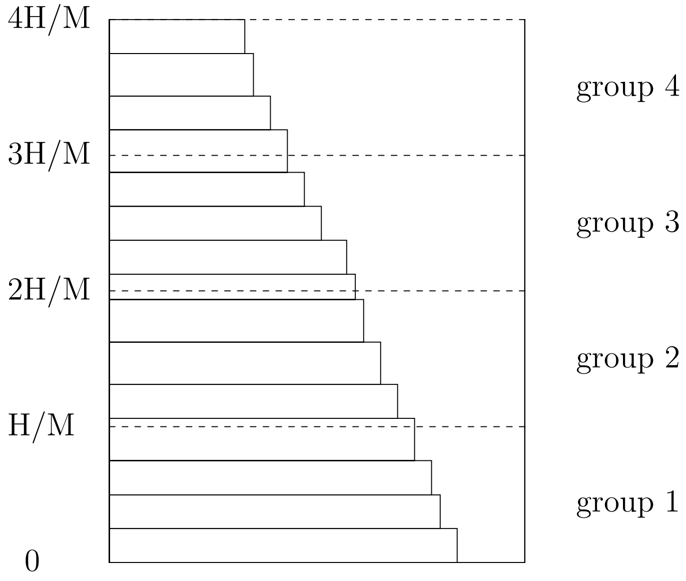

Fig. 12. The stack packing for L_{wide} and partition into groups

$O(1/\epsilon^2)$; otherwise we can neglect the wide rectangles (i.e. use the standard next fit decreasing height (NFDH) algorithm for the narrow rectangles and place the wide rectangles of length at most $O(1/\epsilon^2)$ at the end. We specify M rectangles on the stack as threshold rectangles. A rectangle packed on the stack is called a threshold rectangle, iff it intersects (either with its interior or at the lower side) with a horizonal line at height $ih(L_{wide})/M$ for $i = 0, \ldots, M-1$. Since the stack height $h(L_{wide}) > O(1/\epsilon^2)$, each rectangle on the stack of height at most 1 can intersect with only one line.

The threshold rectangles are used to separate the wide rectangles into M groups. The i.th group consists of the threshold rectangle at line $ih(L_{wide})/M$ and rectangles that lie completely between the lines $ih(L_{wide})/M$ and $(i+1)h(L_{wide})/M$. The widths of all rectangles in group i are rounded up to the width of the threshold rectangle which has the largest width (see Figure 13). This defines our instance L_{sup}; it contains now of rectangles with only M distinct widths. Furthermore, each rectangle's width is larger than ϵ'. Next we apply the fractional strip packing algorithm on the instance L_{sup} and construct an integral nice packing for L_{sup}.

Afterwards, we have to add the rectangles in L_{narrow}. This can be done as follows. First we order these rectangles by non-increasing heights. We place the narrow rectangles in the M' free rectangular regions $R_1, \ldots, R_{M'}$ (with $R_j = [c_j, 1] \times [\ell_j, \ell_{j+1}]$) according to a modified next fit decreasing height (NFDH) strategy (see also Figure 14). First we use NFDH to place the rectangles into R_1. In this heuristic the rectangles are packed on a sequence of levels. The first level is the bottom line in R_1. Each subsequent level is the horizontal line

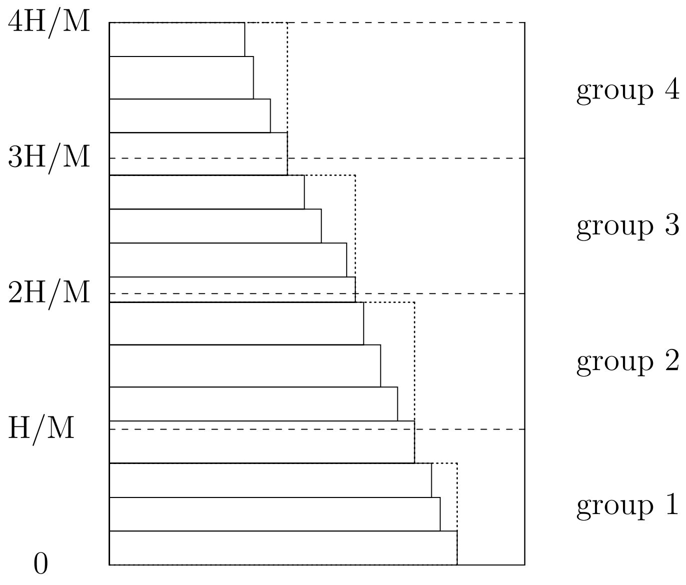

Fig. 13. Linear rounding to construct L_{sup}

above the highest (the leftmost) rectangle on the previous level. The rectangles are placed in a left-justified manner on a level in R_1 until there is not enough space to the right to pack the next rectangle completely on the level. At such a moment, a new level is constructed as above and the packing proceeds on the next level. When a level with the first rectangle does not fit into R_1, then we start a new level at the bottom line of R_2 and use NFDH again. The procedure is iterated until $R_{M'}$ (if necessary). When a level with first rectangle does not fit into $R_{M'}$, then we start a new level at line $\ell_{M'+1}$. In this case we pack the remaining rectangles in the strip with the usual NFDH heuristic. Notice that no narrow rectangle is placed in region R_j (on the first level), if the current narrow rectangle to be placed has width w_i with $w_i + c_j > 1$. In this case the total width c_j of the wide rectangle is at least $1 - \epsilon'$.

The algorithm $STRIP - PACK$ can be summarized as follows:

given: list $L = (r_1, \ldots, r_n)$ of rectangles with heights $h_i, w_i \leq 1$,
(0) set $\epsilon' = \frac{\epsilon}{(2+\epsilon)}$, and $M = (\frac{1}{\epsilon'})^2$,
(1) partition L into L_{narrow} and L_{wide},
(2) construct L_{sup} such that $L_{wide} \leq L_{sup}$ and that L_{sup} has only M distinct widths,
(3) solve the fractional strip packing problem corresponding to L_{sup} approximately using the algorithm by Karp and Karmarkar,
(4) construct an integral nice strip packing for L_{sup},
(5) use modified NFDH for the rectangles in L_{narrow} and pack them into the regions $R_1, \ldots, R_{M'}$ and above if necessary.

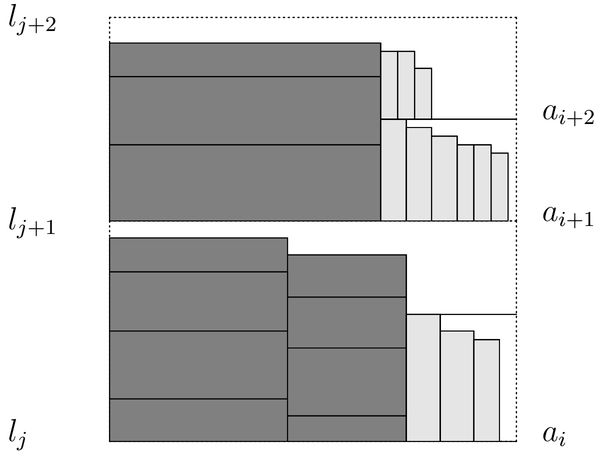

Fig. 14. Adding narrow rectangles by Modified Next Fit Decreasing Height

Let h' be the height after step (4) and let h_{final} be the final height after adding the narrow rectangles. Remember that $M' \leq 2M$. The following Lemmas are useful for the analysis of the algorithm.

Lemma 5.

$$\begin{aligned} LIN(L_{sup}) &\leq LIN(L_{wide})(1 + \tfrac{1}{M\epsilon'}) \\ SIZE(L_{sup}) &\leq SIZE(L_{wide})(1 + \tfrac{1}{M\epsilon'}) \end{aligned}$$

where $LIN(L)$ is the height of the optimal fractional strip packing and $SIZE(L)$ is the total area of all rectangles in L.

Proof. Consider the following generalization of the relation $\leq$. Given any list L of rectangles, first build a stack packing as described above. Let $STACK(L)$ be the polygonal region in the plane covered by the stack packing for L. Then we say $L \leq_g L'$ iff $STACK(L)$ is contained in $STACK(L')$. The relation implies directly that $LIN(L) \leq LIN(L')$ and $SIZE(L) \leq SIZE(L')$.

We define now two sequences L'_{inf} and L'_{sup} of rectangles such that $L'_{inf} \leq_g L_{wide} \leq_g L_{sup} \leq_g L'_{sup}$. To do this we cut the threshold rectangles that intersect a line $ih(L_{wide})/M$ in two pieces and divide the rectangles into M groups of exactly height $h(L_{wide})/M$. Group i consists of all rectangles that lie between line $(i-1)h(L_{wide})/M$ and $ih(L_{wide})/M$. Let a_i, b_i be the smallest and largest width of rectangles in group i. Rounding each rectangle in group i up to b_i (and up to 1 for the first group) generates L'_{sup}. Rounding each rectangle in group i down to b_{i+1} (and down to 0 for the last group) generates L'_{inf}.

Then the $STACK(L'_{sup})$ is equal to the $STACK(L'_{inf})$ plus one rectangle with width 1 and height $h(L_{wide})/M$. This implies that

$$LIN(L'_{sup}) = LIN(L'_{inf}) + h(L_{wide})/M$$

$$SIZE(L'_{sup}) = SIZE(L'_{inf}) + h(L_{wide})/M$$

Therefore, $LIN(L_{sup}) \leq LIN(L'_{sup}) \leq LIN(L'_{inf})+h(L_{wide})/M \leq LIN(L_{wide}) + h(L_{wide})/M$. Since all rectangles in L_{wide} have a width $> \epsilon'$, we have $h(L_{wide})\epsilon' \leq SIZE(L_{wide}) \leq LIN(L_{wide})$. This gives $LIN(L_{sup}) \leq LIN(L_{wide}) (1 + \frac{1}{M\epsilon'})$. In the same way we can prove the bound for $SIZE(L_{sup})$.

Lemma 6. *Let $L_{aux} = L_{sup} \cup L_{narrow}$. If $h_{final} > h'$ then*

$$h_{final} \leq SIZE(L_{aux})/(1-\epsilon') + 4M + 1.$$

Proof. Let $(a_1 < \ldots < a_r)$ be the ordered sequence of levels constructed by modified NFDH, and let $(a_{s_1} < \ldots < a_{s_{r'}})$ be the subsequence of levels with at least one rectangle. Furthermore, let b_{s_i} (b'_{s_i}) be the height of the first (last) narrow rectangle placed on level a_{s_i}, $i = 1, \ldots, r'$. A level a_{s_i} is closed when the next narrow rectangle does not fit completely on the level. Notice that all narrow rectangles on level a_{s_i} have height $\geq b'_{s_i}$. Let $(a_{\bar{s}_1} < \ldots < a_{\bar{s}_{\bar{r}}})$ be the subsequence of $(a_{s_1} < \ldots < a_{s_{r'}})$ such that $a_{\bar{s}_i} + b'_{\bar{s}_i} \leq \ell_{j+1} - 2$ where j is the layer that contains level $a_{\bar{s}_i}$, for $i = 1, \ldots, \bar{r}$. Let $lay(k) = j$ be the layer that contains level a_k. An important fact is that region $R'_j = [0, c_j] \times [\ell_j, \ell_{j+1} - 2]$ is completely covered by wide rectangles in L_{sup}. Let us consider three cases:

Case 1: Level a_k with at least one narrow rectangle and $a_k + b'_k \leq \ell_{lay(k)+1} - 2$. In this case $k = \bar{s}_i$ for $i \in \{1, \ldots, \bar{r}\}$. The modified NFDH algorithm implies that interval $[a_k, a_k + b'_k] \subset [\ell_j, \ell_{j+1} - 2]$. Therefore, an area of at least $b'_{s_i}(1-\epsilon')$ is covered by wide and narrow rectangles in $L_{sup} \cup L_{narrow}$.

Case 2: Level a_k with at least one narrow rectangle and $a_k + b'_k > \ell_{lay(k)+1} - 2$ and $a_k \leq \ell_{lay(k)+1} - 2$. In this case an area of $(\ell_{lay(k)+1} - 2 - a_k)(1-\epsilon')$ is covered by wide and narrow rectangles.

Case 3: Level a_k without any narrow rectangles and $a_k \leq \ell_{lay(k)+1} - 2$. This case may happen, when the wide rectangles have already a total width larger than $1 - \epsilon'$ and the current narrow rectangle to be placed is too wide. In this case an area of at least $(\ell_{lay(k)+1} - 2 - a_k)(1-\epsilon')$ is covered by wide rectangles in L_{sup}.

Notice that case 2 and 3 can happen only once per layer. Let

$$X = \sum_{k \notin \{\bar{s}_1, \ldots, \bar{s}_{\bar{r}}\}, a_k \leq \ell_{lay(k)+1} - 2} (\ell_{lay(k)+1} - 2 - a_k).$$

Then the total size $SIZE(L_{sup} \cup L_{narrow})$ of all wide and narrow rectangles is at least $(1-\epsilon')(X + \sum_{i=1}^{\bar{r}} b'_{\bar{s}_i})$. Using $b'_{\bar{s}_i} \geq b_{\bar{s}_{i+1}}$ for $i = 1, \ldots, \bar{r} - 1$, we get $SIZE(L_{sup} \cup L_{narrow}) \geq (\sum_{i=1}^{\bar{r}-1} b_{\bar{s}_{i+1}} + X)(1-\epsilon')$.

On the other hand, the height of the final packing is at most $\sum_{i=1}^{\bar{r}} b_{\bar{s}_i} + X + 2M'$.

This gives

$$h_{final} \leq \sum_{i=1}^{\bar{r}} b_{\bar{s}_i} + X + 4M$$

$$\le \sum_{i=1}^{\bar{r}-1} b_{\bar{s}_{i+1}} + X + 4M + 1$$

$$\le \frac{SIZE(L_{sup} \cup L_{narrow})}{(1-\epsilon')} + 4M + 1.$$

Let $OPT(L)$ be the minimum height used by an optimal algorithm.

Theorem 1. *Given a list of rectangles $L = (r_1, \ldots, r_n)$ with $w_i, r_i \le 1$, algorithm $STRIP-PACK$ produces a packing of total height*

$$h_{final} \le (1+\epsilon)OPT(L) + O(1/\epsilon^2).$$

The running time of $STRIP-PACK$ is polynomial in n and $1/\epsilon$.

Proof. Lemma 5 implies that

$$SIZE(L_{aux}) \le SIZE(L)(1 + \frac{1}{M\epsilon'}).$$

Suppose that $h_{final} > h'$. In this case using Lemma 6 and $SIZE(L) \le OPT(L)$,

$$h_{final} \le SIZE(L)\frac{(1+\frac{1}{M\epsilon'})}{(1-\epsilon')} + 4M + 1$$

$$\le OPT(L)\frac{(1+\frac{1}{M\epsilon'})}{(1-\epsilon')} + 4M + 1.$$

On the other hand using Lemma 5

$$h' \le LIN(L_{sup}) + 1 + 2M$$

$$\le LIN(L_{wide})(1 + \frac{1}{M\epsilon'}) + 1 + 2M$$

$$\le LIN(L)(1 + \frac{1}{M\epsilon'}) + 1 + 2M$$

$$\le OPT(L)(1 + \frac{1}{M\epsilon'}) + 1 + 2M.$$

Both cases together and the definition of ϵ' and M imply the following bound

$$h_{final} \le OPT(L)\frac{(1+\frac{1}{M\epsilon'})}{(1-\epsilon')} + 4M + 1$$

$$\le (1+\epsilon)OPT(L) + 4(\tfrac{2+\epsilon}{\epsilon})^2 + 1$$

$$\le (1+\epsilon)OPT(L) + O(\tfrac{1}{\epsilon^2}).$$

5.2 Improving the Running Time of the AFPTAS

We can solve the fractional strip packing problem

$$\begin{aligned} &\text{Min } \sum_{j=1}^{q} x_j \\ &\text{s.t. } \sum_{j=1}^{q} \frac{\alpha_{ij}}{\beta_i} x_j \ge 1, \qquad i = 1, \ldots, M, \\ &\qquad x_j \ge 0, \qquad\qquad j = 1, \ldots, q. \end{aligned} \tag{6}$$

by binary search on the optimum value and testing in each step the feasibility of a system of (in-) equalities for a given $r \in [L, U]$ where L, U are lower and upper bounds on the optimum height. A simple upper bound can be computed using approximation algorithms for strip packing or nonmalleable scheduling (see [7,33]). Interestingly, the height of the packing can be bounded in these algorithms by $2\sum_{i=1}^{M} \beta_i w'_i + h_{max}$ where $h_{max} = \max_{R_i \in L_{wide}} h_i$ is the maximum height over all wide rectangles. Notice that the total area $\sum_{i=1}^{M} \beta_i w'_i$ and h_{max} are both lower bounds on the optimum fractional strip packing. Therefore we can use $L = \max\{\sum_{i=1}^{M} \beta_i w'_i, h_{max}\}$ and $U = 3L$.

The system of (in-)equalities is given as follows:

$$\sum_{j=1}^{q} \frac{\alpha_{ij}}{\beta_i} x_j \geq 1,\ i = 1, \ldots, M,\ (x_j) \in B,$$

where

$$B = \{(x_j) | \sum_{j=1}^{q} x_j = r, x_j \geq 0\}.$$

Let us consider the following problem:

$$\lambda^* = \text{Max}\{\lambda | \sum_{j=1}^{q} \frac{\alpha_{ij}}{\beta_i} x_j \geq \lambda,\ i = 1, \ldots, M,\ (x_j) \in B\}. \tag{7}$$

This is a max-min resource sharing problem with one block B and M covering constraints. Let the M coupling (covering) constraints be represented by $Ax \geq \lambda$. We can compute an $(1 - \bar{\epsilon})$ - approximate solution for the max-min resource sharing problem (7) where $\bar{\epsilon} = \Theta(\epsilon)$ in $O(M(\bar{\epsilon}^{-2} + \ln M))$ iterations (see Section 3), each requiring for a given price vector $y = (y_1, \ldots, y_M)$ an $(1 - \bar{\epsilon}/6)$ approximate solution of the block problem

$$\text{Max}\{y^T Ax | x \in B\}. \tag{8}$$

Since B is just a simplex, the optimum of this linear program is attained at a vertex $\tilde{x}$ of B that corresponds to a single configuration C_j with $\tilde{x}_j = r$ and $\tilde{x}_{j'} = 0$ for $j \neq j'$. Thus it is sufficient to find a subset of rectangles that can occur at the same time and has largest associated profit value d_j in the profit vector $d^T = y^T A$. This can be described by an integer linear program with variables z_i that give the number of rectangles of width w'_i. We have to solve approximately for a given M - vector $(y_1, \ldots, y_M)$:

$$\begin{aligned} &\text{Max } \textstyle\sum_{i=1}^{M} \frac{y_i}{\beta_i} z_i \\ &s.t.\ \ \textstyle\sum_{i=1}^{M} w'_i z_i \leq 1 \\ &\qquad z_i \geq 0, integer. \end{aligned} \tag{9}$$

This is a classical Knapsack problem with unbounded variables. This problem can be solved approximately in $O(M + (1/\bar{\epsilon})^3)$ time [27]. The overhead (i.e. the numerical calculations) per iteration is $O(M \ln\ln(M\bar{\epsilon}^{-1}))$. This implies that the max-min resource sharing problem can be solved approximately with value $(1-\bar{\epsilon})\lambda^*$. In our case we have $\lambda^* \geq 1$ for $r \geq OPT$.

There are two possible outcomes of the max-min resource sharing procedure: either we find a solution that covers at least $\sum_j \frac{\alpha_{ij}}{\beta_i} x_j \geq (1-\bar{\epsilon})$ or we can conclude that there is no solution with $\sum_j \frac{\alpha_{ij}}{\beta_i} x_j \geq 1$. In the first case we can choose in the binary search a smaller value for r, and in the second case $OPT > r$ and we must choose a higher value for r. By binary search on r in the interval $[L, 3L]$ we can compute in this way a solution (x_j) with $\sum_{j=1}^q x_j \leq (1+\delta/4)OPT$ and $\sum_{j=1}^q (\alpha_{ij}/\beta_i)x_j \geq (1-\bar{\epsilon})$ where OPT is the height of an optimal fractional strip packing. Now we can set $\tilde{x}_j = x_j(1+4\bar{\epsilon})$ and obtain

$$\sum_{j=1}^{q} \frac{\alpha_{ij}}{\beta_i} \tilde{x}_j \geq (1-\bar{\epsilon})(1+4\bar{\epsilon}) \geq 1$$

for $\bar{\epsilon} \leq 3/4$. In this case, the length of the generated strip packing is at most

$$\sum_{j=1}^{q} \tilde{x}_j \leq OPT(1+4\bar{\epsilon})(1+\delta/4) \leq (1+\delta)OPT$$

by choosing $\delta \leq 1$ and $\bar{\epsilon} = 3\delta/20$. Since the optimum of (6) lies within the interval $[L, 3L]$, the overall running time of the algorithm is

$$O(M(\frac{1}{\delta^2} + \ln M)\ln(\frac{1}{\delta})\max(M + \frac{1}{\delta^3}, M\ln\ln(\frac{M}{\delta}))).$$

Interestingly, the binary search can be avoided by computing only a solution for one value for r (see [23]). This gives a total running time of

$$O(M(\frac{1}{\delta^2} + \ln M)\max(M + \frac{1}{\delta^3}, M\ln\ln(\frac{M}{\delta}))).$$

Using $M = O(1/\epsilon^2)$ and $\delta = \Theta(\epsilon)$, the block solver runs in $O(\epsilon^{-3})$ time and the fractional strip packing can be solved in time $O(\epsilon^{-7})$.

The algorithm above is based on ideas by Plotkin et al. [30] for bin packing (with a more involved and complicated block optimization) and by Jansen and Porkolab for general covering problems [21]. Comparing with the approach in [30], the knapsack problem above has unbounded variables. Plotkin et al. [30] used a decomposition method to reduce the width ρ in the fractional covering problem. Using this technique, the block problem becomes more complicated. In fact the block problem is a knapsack problem with additional constraints on the variables and the underlying approximation scheme has a slower running time of $O(\min(M\epsilon^{-2}, M\log\epsilon^{-1} + \epsilon^{-4})) = O(\epsilon^{-4})$. The number of iterations in their approach is $O(M\log^2\rho(\log^2 M + \epsilon^{-2}\log(M\epsilon^{-1})))$ where ρ is the width of fractional covering problem. The width can be bounded by the optimum value of the fractional strip packing instance or the number of wide rectangles. This would give a slower running time of $O(\epsilon^{-8}\ln(\epsilon^{-3})\ln^2(n))$.

The number of configurations in the computed solution using the max-min resource sharing framework is $O(M(\epsilon^{-2}+\ln M)) = O(\epsilon^{-4})$, since we add one configuration $x_j > 0$ in each iteration. This number can be reduced to $O(\epsilon^{-2})$ by solving several systems of $(M+1)$ linear equalities with $M+2$ variables. Consider a solution $\tilde{x} = (\tilde{x}_j)$ of the linear system:

$$\begin{aligned} &\textstyle\sum_{j=1}^{q} \alpha_{ij}\tilde{x}_j = b_i \geq \beta_i && i = 1, \ldots, M \\ &\textstyle\sum_{j=1}^{q} \tilde{x}_j \quad\; = b_{M+1} \leq (1+\delta)OPT \end{aligned}$$

where w.l.o.g. $\tilde{x}_j > 0$ for $j = 1, \ldots, q'$ and $\tilde{x}_j = 0$ for $j = q'+1, \ldots, q$. Take now a submatrix B with $M+2$ columns and $M+1$ rows and solve the system $Bz = 0$. Then there is a nontrivial vector $\hat{z} \neq 0$ such that $B\hat{z} = 0$. We can eliminate one positive variable $\tilde{x}_j$ corresponding to one of the $M+2$ columns in B (i.e. by using $\tilde{x} = \tilde{x} + \tilde{z}\delta$ where $\tilde{z}$ is the vector of dimension q that contains $\hat{z}$ and additional zeros and δ is appropriate chosen). Each round for eliminating one component can be done in $O(\mathcal{M}(M))$ time, where $\mathcal{M}(n)$ is the time to solve a system of equalities with n variables and n constraints. The number of rounds is $O(\epsilon^{-4})$. This gives $O(\epsilon^{-4}\mathcal{M}(\epsilon^{-2}))$ time to reduce the number of configurations.

6 Analysis for Min-Max Resource Sharing

In this section we give the details for the analysis of the algorithm by Grigoriadis and Khachiyan for the min-max resource sharing problem (for the definitions we refer to Section 2). First we show several properties for the minimum value $\theta(f(x))$ of the potential function Φ_t and for the price vector $p(f(x))$.

Lemma 7.

$$\begin{array}{lll} (a) & p(f(x)) \in P, & \\ (b) & \lambda_{\mathcal{P}}(f(x)) + \frac{t}{M} \;\leq\; \theta(f(x)) \;\leq\; \lambda_{\mathcal{P}}(f(x)) + t, & \\ (c) & p(f(x))^T f(x) \;=\; -t + \theta(f(x)). & \end{array}$$

Proof. By (1) we have $\sum_{m=1}^{M} p_m(f(x)) = 1$ and $p_m(f(x)) \geq 0$, since $\theta(f(x)) > \lambda_{\mathcal{P}}(f(x)) = \max\{f_1(x), \ldots, f_M(x)\} \geq f_m(x)$. This shows (a).

To show (b) we use the fact that $\frac{t}{M}\frac{1}{\theta(f(x))-f_m(x)} \leq 1$ for $m = 1, \ldots, M$. This implies:

$$\begin{aligned} &\tfrac{t}{M}\tfrac{1}{\theta(f(x))-\lambda_{\mathcal{P}}(f(x))} \leq 1 \\ \Longrightarrow\; &\tfrac{t}{M} \leq \theta(f(x)) - \lambda_{\mathcal{P}}(f(x)) \\ \Longrightarrow\; &\lambda_{\mathcal{P}}(f(x)) + \tfrac{t}{M} \leq \theta(f(x)). \end{aligned}$$

For the other inequality we use

$$1 = \frac{t}{M}\sum_{m=1}^{M}\frac{1}{\theta(f(x)) - f_m(x)} \leq \frac{t}{M}\sum_{m=1}^{M}\frac{1}{\theta(f(x)) - \lambda_{\mathcal{P}}(f(x))} = \frac{t}{\theta(f(x)) - \lambda_{\mathcal{P}}(f(x))}.$$

This gives $\theta(f(x)) - \lambda_{\mathcal{P}}(f(x)) \leq t$ or equivalent $\theta(f(x)) \leq \lambda_{\mathcal{P}}(f(x)) + t$.

Finally, (c) is proved by

$$\begin{aligned} p(f(x))^T f(x) &= \tfrac{t}{M}\textstyle\sum_{m=1}^{M}\frac{f_m(x)}{\theta(f(x)) - f_m(x)} \\ &= \tfrac{t}{M}\textstyle\sum_{m=1}^{M}(-1 + \frac{\theta(f(x))}{\theta(f(x)) - f_m(x)}) \\ &= -t + \tfrac{\theta(f(x))t}{M}\textstyle\sum_{m=1}^{M}\frac{1}{\theta(f(x)) - f_m(x)} \\ &= -t + \theta(f(x))\textstyle\sum_{m=1}^{M} p_m(f(x)) = -t + \theta(f(x)). \end{aligned}$$

For small $t > 0$, the minimum value $\theta(f(x))$ approximates the reduced potential value $\phi_t(f(x)) = \Phi_t(\theta(f(x)), f(x))$ as follows.

Lemma 8.

(a) *$\theta(f(x)) - t\ln\theta(f(x)) \leq \phi_t(f(x)) \leq \theta(f(x)) - t\ln t$ for any $t \geq 0$,*
(b) *$\theta(f(x)) \leq (\phi_t(f(x)) - t)/(1-t)$ for any $t \in (0,1)$.*

Proof. Using the definition of $\Phi_t(\theta(f(x)), f(x))$ we get

$$\phi_t(f(x)) = \theta(f(x)) - \frac{t}{M}\sum_{m=1}^{M}\ln(\theta(f(x)) - f_m(x)).$$

Suppose that $\theta = \theta(f(x))$. Since the functions f_m are non-negative,

$$\begin{aligned} \theta &= \phi_t(f(x)) + \tfrac{t}{M}\textstyle\sum_{m=1}^{M}\ln(\theta - f_m(x)) \\ &\leq \phi_t(f(x)) + \tfrac{t}{M}\textstyle\sum_{m=1}^{M}\ln\theta \\ &= \phi_t(f(x)) + t\ln\theta. \end{aligned}$$

This shows the first inequality in (a). To show the second inequality, we use the following inequality of the harmonic, geometric and arithmetic mean [1]. Let $a_1, \ldots, a_M$ be positive real numbers, then $\frac{1}{M}\sum_{m=1}^{M} a_m \geq (\prod_{m=1}^{M} a_m)^{1/M}$. By (1) we obtain

$$1 = \frac{t}{M}\sum_{m=1}^{M}\frac{1}{\theta - f_m(x)} \geq t(\prod_{m=1}^{M}\frac{1}{\theta - f_m(x)})^{1/M}.$$

This inequality can be transformed into:

$$\begin{aligned}
&\tfrac{1}{t} \geq (\textstyle\prod_{m=1}^{M} \tfrac{1}{\theta - f_m(x)})^{1/M} \\
&\Longrightarrow \ln(\tfrac{1}{t}) \geq \tfrac{1}{M} \textstyle\sum_{m=1}^{M} \ln(\tfrac{1}{\theta - f_m(x)}) \\
&\Longrightarrow -\ln t \geq -\tfrac{1}{M} \textstyle\sum_{m=1}^{M} \ln(\theta - f_m(x)) \\
&\Longrightarrow 0 \geq \ln t - \tfrac{1}{M} \textstyle\sum_{m=1}^{M} \ln(\theta - f_m(x)) \\
&\Longrightarrow 0 \geq t \ln t - \tfrac{t}{M} \textstyle\sum_{m=1}^{M} \ln(\theta - f_m(x)) \\
&\Longrightarrow 0 \geq t \ln t + \phi_t(f(x)) - \theta \\
&\Longrightarrow \phi_t(f(x)) \leq \theta - t \ln t.
\end{aligned}$$

The inequality in (b) can be proven as follows. Using $\ln \theta \leq \theta - 1$ and (a) we get:

$$\theta - t(\theta - 1) \leq \theta - t \ln \theta \leq \phi_t(f(x)).$$

The property above will be used to bound the difference between two reduced potential values. In the next part, we study algorithm A_L. First we prove that if the initial vector x_0 satisfies $\lambda_{\mathcal{P}}(f(x_0)) \leq 1$ and the stopping criteria $p(f(x))^T f(\hat{x}) \geq \theta(f(x)) - 2t$ is fulfilled, then the pair $x, p(f(x))$ is an approximate solution of the primal and dual problem with absolute error σ.

Lemma 9. *For a given $x \in B$, let $p(f(x)) \in P$ with $p_m(f(x)) = \frac{t}{M} \frac{1}{\theta(f(x)) - f_m(x)}$ and $\hat{x}$ computed by $ABS(p(f(x)), \frac{t}{3})$ where $t = 3\sigma/7$. Let $\lambda_{\mathcal{P}}(f(x_0)) \leq 1$, and $\sigma \in (0, 1/3)$. If $p(f(x))^T f(\hat{x}) \geq \theta(f(x)) - 2t$, then*

$$\begin{aligned}
\lambda_{\mathcal{P}}(f(x)) &< \lambda_{\mathcal{P}}^* + \sigma, \\
\Lambda(p(f(x))) &> \lambda_{\mathcal{P}}^* - \sigma.
\end{aligned}$$

Proof. Let x^* be an optimal solution of problem $(\mathcal{P})$ with $\lambda_{\mathcal{P}}(f(x^*)) = \lambda_{\mathcal{P}}^*$. The block solver generates a solution $\hat{x}$ such that

$$\begin{aligned}
p(f(x))^T f(\hat{x}) &\leq (1 + t/3)\Lambda(p(f(x))) = (1 + \sigma/7)\Lambda(p(f(x))) \\
&= (1 + \sigma/7) \min\{p(f(x))^T f(y) | y \in B\} \leq (1 + \sigma/7) p(f(x))^T f(x^*) \\
&= (1 + \sigma/7) \textstyle\sum_{m=1}^{M} p_m(f(x)) f_m(x^*) \\
&\leq (1 + \sigma/7) \max_{1 \leq \ell \leq M} f_\ell(x^*) \textstyle\sum_{m=1}^{M} p_m(f(x)) \\
&= (1 + \sigma/7) \lambda_{\mathcal{P}}(f(x^*)) \leq \lambda_{\mathcal{P}}^* + (\sigma/7) \lambda_{\mathcal{P}}(f(x_0)) \\
&\leq \lambda_{\mathcal{P}}^* + \sigma/7.
\end{aligned}$$

If $p(f(x))^T f(\hat{x}) \geq \theta(f(x)) - 2t$ then $\lambda^*_{\mathcal{P}} + \sigma/7 \geq \theta(f(x)) - 2t$. By Lemma 7 (b) we have $\theta(f(x)) > \lambda_{\mathcal{P}}(f(x))$. Therefore, $\lambda_{\mathcal{P}}(f(x)) < \lambda^*_{\mathcal{P}} + \sigma/7 + 2t = \lambda^*_{\mathcal{P}} + \sigma$ (using $t = 3\sigma/7$).

Furthermore,

$$\begin{aligned} \Lambda(p(f(x))) &\geq \tfrac{p(f(x))^T f(\hat{x})}{1+\sigma/7} \\ &\geq \tfrac{\theta(f(x))-2t}{1+\sigma/7} \\ &> \tfrac{\lambda_{\mathcal{P}}(f(x))-6\sigma/7}{1+\sigma/7} \\ &\geq \tfrac{\lambda^*_{\mathcal{P}}-6\sigma/7}{1+\sigma/7} \\ &\geq \lambda^*_{\mathcal{P}} - \sigma. \end{aligned}$$

The last inequality follows, since

$$\begin{aligned} &\lambda^*_{\mathcal{P}} - 6\sigma/7 \geq (1+\sigma/7)(\lambda^*_{\mathcal{P}} - \sigma) \\ \iff &\lambda^*_{\mathcal{P}} - 6\sigma/7 \geq \lambda^*_{\mathcal{P}} + \sigma\lambda^*_{\mathcal{P}}/7 - \sigma - \sigma^2/7 \\ \iff &\sigma/7 + \sigma^2/7 \geq \sigma\lambda^*_{\mathcal{P}}/7 \\ \iff &1 + \sigma \geq \lambda^*_{\mathcal{P}}. \end{aligned}$$

In last step we use $\lambda^*_{\mathcal{P}} \leq \lambda_{\mathcal{P}}(f(x_0)) \leq 1$ and $\sigma \geq 0$.

In order to bound the number of iterations of algorithm A_L we use the next two Lemmas.

Lemma 10. *Suppose that $\phi_t(f(x)) \leq \phi_t(f(x_0))$ for a vector x computed by algorithm A_L and an initial vector x_0 with $\lambda_{\mathcal{P}}(f(x_0)) \leq 1$. Then $p(f(x))^T(f(x) + f(\hat{x})) \leq 5/2$.*

Proof. First we have $p(f(x))^T f(\hat{x}) \leq \lambda^*_{\mathcal{P}} + \sigma/7 \leq 1 + \sigma/7 \leq 1 + 1/21 = 22/21$ using Lemma 9 and $\sigma \leq 1/3$. Therefore it is sufficient to show that $p(f(x))^T f(x) \leq 5/2 - 22/21 = 61/42$. Since $p(f(x))^T f(x) = -t + \theta(f(x))$ (see Lemma 7), let us test whether

$$\theta(f(x)) - t \leq 61/42.$$

By Lemma 8 (b) and the assumption $\phi_t(f(x)) \leq \phi_t(f(x_0))$,

$$\theta(f(x)) \leq \frac{\phi_t(f(x)) - t}{1-t} \leq \frac{\phi_t(f(x_0)) - t}{1-t}.$$

Furthermore, Lemma 8 (a) implies $\phi_t(f(x_0)) \leq \theta(f(x_0)) - t \ln t$. Since $\theta(f(x_0)) \leq \lambda_{\mathcal{P}}(f(x_0)) + t$ (by Lemma 7 (b)) and $\lambda_{\mathcal{P}}(f(x_0)) \leq 1$,

$$\theta(f(x)) \leq \frac{\lambda_{\mathcal{P}}(f(x_0)) - t\ln t}{1-t} \leq \frac{1 - t\ln t}{1-t}.$$

This is equivalent to:

$$\begin{aligned}\theta(f(x)) - t &\le \tfrac{1-t\ln t}{1-t} - \tfrac{t(1-t)}{1-t}\\ &= \tfrac{1-t}{1-t} + \tfrac{t^2-t\ln t}{1-t}\\ &= 1 + \tfrac{t^2-t\ln t}{1-t}.\end{aligned}$$

The function $g(t) = \frac{t^2-t\ln t}{1-t}$ is monotonically increasing for $t \in (0,1)$. Moreover, A_L uses only values $t = 3\sigma/7 \le 1/7$. This implies that

$$\theta(f(x)) - t \le 1 + \frac{(1/7)^2 - (1/7)\ln(1/7)}{1-1/7} \le 61/42.$$

Lemma 11. *For any two consecutive iterations, the vectors x, x' computed by algorithm A_L are such that*

$$\phi_t(f(x')) \le \phi_t(f(x)) - \frac{t^3}{13M}.$$

Proof. Let $x' = (1-\tau)x + \tau\hat{x}$. Suppose that $\theta = \theta(f(x))$. Since f_m is convex and using the definition of $p_m(f(x))$,

$$\begin{aligned}\theta - f_m(x') &= \theta - f_m((1-\tau)x + \tau\hat{x})\\ &\ge \theta - (1-\tau)f_m(x) - \tau f_m(\hat{x})\\ &= (\theta - f_m(x))(1 + \tau\tfrac{f_m(x)-f_m(\hat{x})}{\theta - f_m(x)})\\ &= (\theta - f_m(x))(1 + \tfrac{\tau M}{t}p_m(f(x))(f_m(x) - f_m(\hat{x}))).\end{aligned}$$

The last expression (using the fact that f_m is non-negative and $p_m(f(x)) \ge 0$) can be bounded as follows:

$$\begin{aligned}\tfrac{\tau M}{t}p_m(f(x))(f_m(x) - f_m(\hat{x})) &\ge -\tfrac{\tau M}{t}p_m(f(x))f_m(\hat{x}) \ge -\tfrac{\tau M}{t}p(f(x))^T f(\hat{x})\\ &\ge -(\lambda^*_{\mathcal{P}} + \sigma/7)\tfrac{\tau M}{t} \ge -\tfrac{22}{21}\tfrac{\tau M}{t} = -\tfrac{22}{21}\tfrac{4t}{25+10t} > -1.\end{aligned}$$

The last inequality follows by simple math calculation (equivalent to $21 \cdot 25 + 210t > 88t$ for any $t \ge 0$). Let $p_m = p_m(f(x))$ and $p = p(f(x))$. Combining both statements above,

$$\begin{aligned}&\Phi_t(\theta(f(x)), f(x'))\\ &= \theta(f(x)) - \tfrac{t}{M}\textstyle\sum_{m=1}^M \ln(\theta(f(x)) - f_m(x'))\\ &\le \theta(f(x)) - \tfrac{t}{M}\textstyle\sum_{m=1}^M \ln(\theta(f(x)) - f_m(x))\\ &- \tfrac{t}{M}\textstyle\sum_{m=1}^M \ln(1 + \tfrac{\tau M}{t}p_m(f_m(x) - f_m(\hat{x})))\\ &= \phi_t(f(x)) - \tfrac{t}{M}\textstyle\sum_{m=1}^M \ln(1 + \tfrac{\tau M}{t}p_m(f_m(x) - f_m(\hat{x}))).\end{aligned}$$

Since $\Phi_t(\theta', f(x')) = +\infty$ for $\theta' \leq \lambda' = \lambda_{\mathcal{P}}(f(x'))$, we get

$$\begin{aligned}\phi_t(f(x')) &= \Phi_t(\theta(f(x')), f(x')) = \min_{\lambda'<\xi<\infty} \Phi_t(\xi, f(x')) \\ &= \min_{0<\xi'<\infty} \Phi_t(\xi', f(x')) \leq \Phi_t(\theta(f(x)), f(x')) \\ &\leq \phi_t(f(x)) - \tfrac{t}{M} \textstyle\sum_{m=1}^{M} \ln(1 + \tfrac{\tau M}{t} p_m(f_m(x) - f_m(\hat{x}))).\end{aligned}$$

This gives directly the following bound

$$\phi_t(f(x')) - \phi_t(f(x)) \leq -\frac{t}{M}\sum_{m=1}^{M} \ln(1 + \frac{\tau M}{t} p_m(f_m(x) - f_m(\hat{x}))). \qquad (10)$$

By induction on the number of iterations of A_L, one can prove that $\phi_t(f(x)) \leq \phi_t(f(x_0))$ for any vector $x \in B$ computed by algorithm A_L. If the number of iterations is zero, the condition is trivial. In the following we show a stronger inequality $\phi_t(f(x')) \leq \phi_t(f(x)) - \frac{t^3}{13M}$ for the next vector x' computed by A_L. Using this inequality, we get directly $\phi_t(f(x')) < \phi_t(f(x)) \leq \phi_t(f(x_0))$.

To prove the stronger inequality, let us rewrite the right expression in (10) as $L + R$ and use Taylor's formula for $\ln(1+x) = \sum_{i=1}^{\infty}(-1)^{i+1}\frac{x^i}{i}$ to get:

$$\begin{aligned}L &= -\tau \textstyle\sum_{m=1}^{M} p_m(f_m(x) - f_m(\hat{x})) \\ &= -\tau p^T(f(x) - f(\hat{x})), \\ R &= \tfrac{t}{M}\textstyle\sum_{i=2}^{\infty} \tfrac{1}{i}(\tfrac{-\tau M}{t})^i \textstyle\sum_{m=1}^{M}(p_m(f_m(x) - f_m(\hat{x})))^i \\ &\leq \tfrac{t}{M}\textstyle\sum_{i=2}^{\infty} |\tfrac{1}{i}(\tfrac{-\tau M}{t})^i \textstyle\sum_{m=1}^{M}(p_m(f_m(x) - f_m(\hat{x})))^i| \\ &\leq \tfrac{t}{M}\textstyle\sum_{i=2}^{\infty} \tfrac{1}{i}(\tfrac{\tau M}{t})^i \textstyle\sum_{m=1}^{M}(p_m(f_m(x) + f_m(\hat{x})))^i \\ &\leq \tfrac{t}{M}\textstyle\sum_{i=2}^{\infty} \tfrac{1}{i}(\tfrac{\tau M}{t} \textstyle\sum_{m=1}^{M}(p_m(f_m(x) + f_m(\hat{x}))))^i \\ &\leq \tfrac{t}{M}\textstyle\sum_{i=2}^{\infty} \tfrac{1}{i}((\tfrac{\tau M}{t}) p^T(f(x) + f(\hat{x})))^i\end{aligned}$$

where $p_m = p_m(f(x))$ and $p = p(f(x))$. Using $\phi_t(f(x)) \leq \phi_t(f(x_0))$ by induction for x and Lemma 10, we get $p^T(f(x) + f(\hat{x})) \leq 5/2$ which can be used to bound

$$R \leq \frac{t}{M}\sum_{i=2}^{\infty} \frac{1}{i}(\frac{5\tau M}{2t})^i \leq -\frac{t}{M}[\ln(1 - \frac{5\tau M}{2t}) + \frac{5\tau M}{2t}].$$

Since x does not satisfy the stopping criteria, $p(f(x))^T f(\hat{x}) < \theta(f(x)) - 2t$. Using Lemma 7 (c), $p(f(x))^T(f(x) - f(\hat{x})) = \theta(f(x)) - t - p(f(x))^T f(\hat{x}) > t$. This implies that $L < -\tau t$. Combining the bounds for L and R

$$\phi_t(f(x')) - \phi_t(f(x)) \leq L + R < -t[\tau + \frac{1}{M}[\ln(1 - \frac{5\tau M}{2t}) + \frac{5\tau M}{2t}]].$$

Inserting the maximum $\tau = \frac{4t^2}{M(10t+25)}$ and math calculation we get

$$\phi_t(f(x')) - \phi_t(f(x)) < -\frac{t}{M}[\frac{2t}{5} - \ln(1 + 2t/5)].$$

Let $g(u) = (u - \ln(1+u))/u^2$. The function g is monotonically decreasing for $u > 0$. In our case $0 \leq u \leq 2/35$, since $u = 2t/5 \leq 2/35$ for $t \leq 1/7$. For these values $g(u) \geq g(2/35)$. This implies that $(u - \ln(1 + u)) \geq g(2/35)u^2 > \frac{4t^2}{25}g(2/35) > t^2/13$. Thus, $\phi_t(f(x')) - \phi_t(f(x)) < -\frac{t^3}{13M}$. This proves also the induction step for the next vector x' computed by algorithm A_L: $\phi_t(f(x')) \leq \phi_t(f(x_0))$.

The proof above shows that the reduced potential values are decreasing for consecutive iterations. Now we are able to prove the first main result about the number of iterations of algorithm A_L.

Theorem 2. *For any initial $x_0 \in B$ such that $\lambda_{\mathcal{P}}(f(x_0)) \leq 1$ and any $\sigma \in (0, 1/3]$, algorithm A_L halts in*

$$O\big(M\big(\frac{\lambda_{\mathcal{P}}(f(x_0)) - \lambda^*_{\mathcal{P}}}{\sigma^3} + \frac{\ln\frac{1}{\sigma}}{\sigma^2}\big)\big)$$

iterations.

Proof. By Lemma 7 (b), $\theta(f(x_0)) \leq \lambda_{\mathcal{P}}(f(x_0)) + t$ and by Lemma 8 (a) $\phi_t(f(x_0)) \leq \theta(f(x_0)) - t \ln t \leq \lambda_{\mathcal{P}}(f(x_0)) + t - t \ln t$ For any vector x produced by algorithm A_L (see Lemma 8 (b)),

$$\phi_t(f(x)) \geq t + (1-t)\theta(f(x)) \geq t + (1-t)\lambda_{\mathcal{P}}(f(x)) \geq t + (1-t)\lambda^*_{\mathcal{P}} \geq \lambda^*_{\mathcal{P}}.$$

Combining both inequalities yields

$$\phi_t(f(x_0)) - \phi_t(f(x)) \leq \lambda_{\mathcal{P}}(f(x_0)) - \lambda^*_{\mathcal{P}} + t - t \ln t.$$

Each iteration decreases the reduced potential value $\phi_t(f(x_0))$ by at least $t^3/13M$ (using Lemma 11). Let $f(x_N)$ be the function value after N iterations. Then, $\phi_t(f(x_N)) \leq \phi_t(f(x_0)) - \frac{Nt^3}{13M}$ or equivalent $\phi_t(f(x_0)) - \phi_t(f(x_N)) \geq \frac{Nt^3}{13M}$. Furthermore we have $\phi_t(f(x_0)) - \phi_t(f(x_N)) \leq \lambda_{\mathcal{P}}(f(x_0)) - \lambda^*_{\mathcal{P}} + t - t \ln t$. These inequalities imply $\lambda_{\mathcal{P}}(f(x_0)) - \lambda^*_{\mathcal{P}} + t + t \ln(1/t) \geq Nt^3/(13M)$. In other words the number N of iterations is bounded by

$$O(M(\frac{\lambda_{\mathcal{P}}(f(x_0)) - \lambda^*_{\mathcal{P}}}{t^3} + \frac{1 + \ln(1/t)}{t^2})) = O(M(\frac{\lambda_{\mathcal{P}}(f(x_0)) - \lambda^*_{\mathcal{P}}}{\sigma^3} + \frac{\ln(1/\sigma)}{\sigma^2})).$$

Remark. The stopping criteria is not used directly for vector x_N. But we use the fact that if the criteria is not fulfilled then the reduced potential value is decreased (see proof of Lemma 11).

In the next theorem we study the number of iterations of the ternary search (described in Section 2).

Theorem 3. *For any initial point $x_0 \in B$ and any accuracy $\epsilon \in (0,1)$, the ternary search TS procedure halts in*

$$O(M(\epsilon^{-2}\ln\epsilon^{-1} + \ln(\lambda_{\mathcal{P}}(f(x_0))/\lambda^*_{\mathcal{P}})))$$

iterations with an $x \in B$ such that $\lambda_{\mathcal{P}}(f(x)) \le (1+\epsilon)\lambda^*_{\mathcal{P}}$ and a price vector $p \in P$ such that $\Lambda(p) = \min\{p^T f(x) | x \in B\} \ge (1-\epsilon)\lambda^*_{\mathcal{P}}$.

Proof. First, the length of the interval is decreased by a factor of 2/3 in each iteration. To bound the running time of A_L, we notice that

$$\frac{v\lambda_{\mathcal{P}}(f(x)) - v\lambda^*_{\mathcal{P}}}{\sigma} \le \frac{v(\overline{\lambda} - \lambda^*_{\mathcal{P}})}{\sigma} \le \frac{v\delta}{\sigma} = 3.$$

Since A_L works with the scaled function vf, this implies that the second term $O(\sigma^{-2}\ln\sigma^{-1})$ in the number of iterations of algorithm A_L always dominates the first term (that can be bounded by $O(\sigma^{-2})$). In other words the number of iterations in A_L is at most $O(M\sigma^{-2}\ln\sigma^{-1})$ where $\sigma \le 1/3$. To bound the total number of iterations we count the number of iterations of the *repeat* loop with $\underline{\lambda} = 0$ and $\underline{\lambda} > 0$.

Case 1: Let us calculate the number of iterations with $\underline{\lambda} = 0$. In this case the ending condition $\delta \le 3\epsilon\underline{\lambda}$ is never satisfied and $\sigma = 1/3$. Consider N steps where $\overline{\lambda}$ is replaced in each step (and $\underline{\lambda}$ is unchanged). This is possible only if $\lambda_{\mathcal{P}}(f(x_0))(2/3)^N \ge \lambda^*_{\mathcal{P}}$. This is equivalent to $\ln(\lambda_{\mathcal{P}}(f(x_0))/\lambda^*_{\mathcal{P}}) \ge N\ln(3/2)$ and shows that $N \le O(\ln(\lambda_{\mathcal{P}}(f(x_0))/\lambda^*_{\mathcal{P}}))$.

Since $\sigma = 1/3$, A_L needs $O(M)$ steps in each iteration. Therefore, the total number of iterations is $O(M\ln(\lambda_{\mathcal{P}}(f(x_0))/\lambda^*_{\mathcal{P}}))$.

Case 2: Let us calculate here the number of iterations with $\underline{\lambda} > 0$. Let $\delta_0, \delta_1, \ldots$ be the sequence of interval lengths during these iterations. Furthermore, let $\sigma_0, \sigma_1, \ldots$ and $\underline{\lambda}_0, \overline{\lambda}_0, \underline{\lambda}_1, \overline{\lambda}_1, \ldots$ be the other parameters. Then the following properties hold:

(1) $\sigma_i \le \delta_i/\underline{\lambda}_i$,
(2) $\delta_{i+1} = (2/3)\delta_i$,
(3) $\underline{\lambda}_{i+1} \ge \underline{\lambda}_i \ge \ldots \ge \underline{\lambda}_0$,
(4) $\overline{\lambda}_{i+1} \le \overline{\lambda}_i \le \ldots \le \overline{\lambda}_0$.

The algorithm halts when $\delta_N \le 3\epsilon\underline{\lambda}_N$. Using the properties above $\sigma_i \le (2/3)^i\delta_0/\underline{\lambda}_0$ and $\sigma_i \ge (2/3)^i\delta_0/(9\overline{\lambda}_0)$. In the first iteration with $\underline{\lambda}_0 > 0$ we have $\underline{\lambda}_0 = \delta/3$ and $\delta_0 = 2\delta/3$. This gives $\delta_0/\underline{\lambda}_0 = 2$ and $\delta_0/\overline{\lambda}_0 = 2/3$. In the worst case $\underline{\lambda}_0$ is not modified. This implies that $\delta_N = (2/3)^N\delta_0 \le 3\epsilon\underline{\lambda}_0$. In other words at most $N = O(\log(1/\epsilon))$ steps are possible.

The total number of iterations in A_L is

$$O(M\sum_{i=1}^{\log(1/\epsilon)} \frac{\ln\sigma_i^{-1}}{(\sigma_i)^2}) = O(M\sum_{i=1}^{\log(1/\epsilon)} \frac{\ln(3/2)^i}{((2/3)^i)^2}) = O(M\sum_{i=1}^{\log(1/\epsilon)} i((3/2)^i)^2)$$

that can be bounded by $O(M\sum_{i=1}^{\log(1/\epsilon)} i(2^i)^2)$.

All terms $O(Mi(2^i)^2)$ are dominated by the last term for $i = \log(1/\epsilon)$. Thus, this number is bounded by $O(M\log(1/\epsilon)(1/\epsilon)^2) = O(M\ln(1/\epsilon)(1/\epsilon)^2)$. Therefore, the total number of iterations (the number of block optimizations) is at most $O(M(\epsilon^{-2}\ln\epsilon^{-1} + \ln(\lambda_{\mathcal{P}}(f(x_0))/\lambda^*_{\mathcal{P}})))$.

The algorithm halts when $\delta_N \leq 3\epsilon\underline{\lambda}_N$; and the last computed x satisfies $\lambda_{\mathcal{P}}(f(x)) \leq \lambda^*_{\mathcal{P}} + \delta_N/3$. Since $\underline{\lambda}_N \leq \lambda^*_{\mathcal{P}}$, we have $\lambda_{\mathcal{P}}(f(x)) \leq \lambda^*_{\mathcal{P}} + \delta_N/3 \leq (1+\epsilon)\lambda^*_{\mathcal{P}}$. The last computed price vector p satisfies also $\Lambda(p) \geq \lambda^*_{\mathcal{P}} - \delta_N/3 \geq \lambda^*_{\mathcal{P}} - \epsilon\underline{\lambda}_N \geq (1-\epsilon)\lambda^*_{\mathcal{P}}$.

An initial solution x_0 can be computed by one block optimization step ABS $(e',1)$ where $e' = (1/M,\ldots,1/M)^T$. The next Lemma implies that we can always assume that $\lambda_{\mathcal{P}}(f(x_0)) \leq 2M\lambda^*_{\mathcal{P}}$.

Lemma 12.

$$\lambda_{\mathcal{P}}(f(x_0)) \leq 2M\lambda^*_{\mathcal{P}}.$$

Proof. The block solver $ABS(e',1)$ generates a solution x_0 such that $(e')^T f(x_0) \leq (1+1)\lambda_{\mathcal{P}}(e')$. This implies $(e')^T f(x_0) = \sum_{m=1}^M f_m(x_0)/M \leq 2\Lambda(e')$ and can be transformed into $f_\ell(x_0) \leq \sum_{m=1}^M f_m(x_0) \leq 2M\Lambda(e') \leq 2M\lambda^*_{\mathcal{P}}$ for each $\ell = 1,\ldots,M$. The last inequality $\Lambda(p) \leq \lambda^*_{\mathcal{P}}$ holds for any price vector $p \in P$. Since $\lambda_{\mathcal{P}}(f(x)) = \max\{f_1(x),\ldots,f_M(x)\}$, we get $\lambda_{\mathcal{P}}(f(x_0)) \leq 2M\lambda^*_{\mathcal{P}}$.

Corollary 1. *For any accuracy $\epsilon > 0$, the algorithm TS halts in*

$$O(M(\epsilon^{-2}\ln\epsilon^{-1} + \ln M))$$

iterations.

Remark. In the analysis above we have assumed that the root $\theta(f(x))$ can be computed exactly. Computing $\theta(f(x))$ up to an accuracy of $O(\ln(M/\epsilon))$ bits (or an absolute error of $O(\epsilon^2/M)$) guarantees that the approximate price vector $\tilde{p}$ has only a relative error of $(1-\delta)p \leq \tilde{p} \leq (1+\delta)p$ and we lose at most a multiplicative factor of $(1+\delta)$ in the block optimization. Such an error in the block optimization can be neglected. Therefore, the bound on the number of iterations in Corollary 1 remains valid.

Since $\Phi_t(\theta, f(x))$ is convex, its minimum $\theta(f(x))$ can be computed to this accuracy by binary search in the interval $[\lambda_{\mathcal{P}}(f(x)) + t/M, \lambda_{\mathcal{P}}(f(x)) + t]$. This requires $O(\ln(M/\epsilon))$ many binary search steps. Since each sum $\sum_{m=1}^M 1/(\theta - f_m(x))$ can be computed in $O(M)$ time, this gives $O(M\ln(M/\epsilon))$ arithmetic operations per iteration. This bound can be further improved by using Newton's method to $O(M\ln\ln(M/\epsilon))$.

7 Analysis for Max-Min Resource Sharing

In this section we show the details for the max-min resource sharing problem (for the definitions we refer to Section 2). First, we prove the following properties for the price vector $p(f(x)$, the objective function value $\lambda_{\mathcal{C}}(f(x))$ and the maximizer $\theta(f(x))$:

Lemma 13.

(a) $p(f(x)) \in P$,
(b) $\frac{\lambda_{\mathcal{C}}(f(x))}{1+t} \le \theta(f(x)) \le \frac{\lambda_{\mathcal{C}}(f(x))}{1+t/M}$,
(c) $p(f(x))^T f(x) = (1+t)\theta(f(x))$.

Proof. By (3) and (4), we have $\sum_{m=1}^M p_m(f(x)) = \sum_{m=1}^M \frac{t\theta(f(x))}{M} \frac{1}{f_m(x)-\theta(f(x))} = 1$ and $p_m(f(x)) \ge 0$ for $0 \le \theta(f(x)) < \lambda_{\mathcal{C}}(f(x))$. In other words $p(f(x)) \in P$. In order to show (b) consider the equality $\sum_{m=1}^M \frac{1}{f_m(x)-\theta(f(x))} = \frac{M}{t\theta(f(x))}$. Since $\lambda_{\mathcal{C}}(f(x)) \le f_m(x)$, we have $f_m(x) - \theta(f(x)) \ge \lambda_{\mathcal{C}}(f(x)) - \theta(f(x))$ or equivalent $1/(f_m(x) - \theta(f(x))) \le 1/(\lambda_{\mathcal{C}}(f(x)) - \theta(f(x)))$. This inequality and the equality above imply

$$\sum_{m=1}^M \frac{1}{\lambda_{\mathcal{C}}(f(x))-\theta(f(x))} \ge \frac{M}{t\theta(f(x))}$$

$$\Longrightarrow \frac{1}{\lambda_{\mathcal{C}}(f(x))-\theta(f(x))} \ge \frac{1}{t\theta(f(x))}$$

$$\Longrightarrow \lambda_{\mathcal{C}}(f(x)) - \theta(f(x)) \le t\theta(f(x))$$

$$\Longrightarrow \lambda_{\mathcal{C}}(f(x)) \le t\theta(f(x)) + \theta(f(x)) = (1+t)\theta(f(x))$$

$$\Longrightarrow \frac{\lambda_{\mathcal{C}}(f(x))}{1+t} \le \theta(f(x)).$$

To show the upper bound for $\theta(f(x))$ we use (3):

$$\frac{1}{\lambda_{\mathcal{C}}(f(x))-\theta(f(x))} \le \frac{M}{t\theta(f(x))}$$

$$\Longrightarrow \lambda_{\mathcal{C}}(f(x))) - \theta(f(x)) \ge \frac{t}{M}\theta(f(x))$$

$$\Longrightarrow \lambda_{\mathcal{C}}(f(x)) \ge (1 + \frac{t}{M})\theta(f(x))$$

$$\Longrightarrow \theta(f(x)) \le \frac{\lambda_{\mathcal{C}}(f(x))}{1+\frac{t}{M}}.$$

This gives the bounds in (b). Furthermore, we obtain for the scalar product

$$p(f(x))^T f(x) = \frac{t\theta(f(x))}{M} \sum_{m=1}^M \frac{f_m(x)}{f_m(x)-\theta(f(x))}$$

$$= \frac{t\theta(f(x))}{M} \sum_{m=1}^M (1 + \frac{\theta(f(x))}{f_m(x)-\theta(f(x))})$$

$$= t\theta(f(x)) + \theta(f(x)) \sum_{m=1}^M \frac{t\theta(f(x))}{M(f_m(x)-\theta(f(x)))}$$

$$= t\theta(f(x)) + \theta(f(x)) \sum_{m=1}^M p_m(f(x)) = \theta(f(x))(t+1).$$

Notice that $\theta(f(x)) < p^T f(\hat{x}) + p^T f(x)$ (by Lemma 13), and therefore the step length

$$\tau = \frac{t\theta(f(x))v(x,\hat{x})}{2M(p^T f(\hat{x}) + p^T f(x))} \le 1$$

where $v(x,\hat{x}) = \frac{p^T f(\hat{x}) - p^T f(x)}{p^T f(\hat{x}) + p^T f(x)} \leq 1$. Furthermore, $v(x,\hat{x}) > t > 0$ implies that $\tau > 0$. Therefore, $\tau \in (0,1]$. The following lemma provides the stopping criteria corresponding to parameter $v(x,\hat{x})$.

Lemma 14. *Suppose $\epsilon \in (0,1)$ and $t = \epsilon/6$. For a given $x \in B$, let $p(f(x)) \in P$ as defined in (4) and $\hat{x}$ computed by $ABS(p(f(x)),t)$. If $v(x,\hat{x}) \leq t$, then the pair x, $p(f(x)))$ solves $(\mathcal{C}_\epsilon)$ and $(\mathcal{D}_\epsilon)$, respectively.*

Proof. First rewrite condition $v(x,\hat{x}) \leq t$ by using (5): $p^T f(\hat{x})(1-t) \leq p^T f(x)$ $(1+t)$ where $p = p(f(x))$. Then use that $p^T f(\hat{x}) \geq (1-t)\Lambda(p)$, $p^T f(x) = (1+t)\theta(f(x))$ and $\theta(f(x)) < \lambda_{\mathcal{C}}(f(x))$ by Lemma 13. This gives

$$\begin{aligned}\Lambda(p) \;\leq\; \tfrac{1}{(1-t)} p^T f(\hat{x}) \;&\leq\; \tfrac{(1+t)}{(1-t)^2} p^T f(x) \;=\; \tfrac{(1+t)^2}{(1-t)^2}\theta(f(x)) \\ &<\; \tfrac{(1+t)^2}{(1-t)^2}\lambda_{\mathcal{C}}(f(x)) \;\leq\; (1+\epsilon)\lambda_{\mathcal{C}}(f(x)).\end{aligned}$$

The last inequality follows from the definition of $t = \epsilon/6$. Using $\lambda^*_{\mathcal{C}} \leq \Lambda(p) \leq (1+\epsilon)\lambda_{\mathcal{C}}(f(x))$, one has $\lambda_{\mathcal{C}}(f(x)) \geq \frac{1}{(1+\epsilon)}\lambda^*_{\mathcal{C}} > (1-\epsilon)\lambda^*_{\mathcal{C}}$ for any $\epsilon > 0$, which gives $(\mathcal{C}_\epsilon)$. Using $\lambda_{\mathcal{C}}(f(x)) \leq \lambda^*_{\mathcal{C}}$, one gets $\Lambda(p) \leq (1+\epsilon)\lambda_{\mathcal{C}}(f(x)) \leq (1+\epsilon)\lambda^*_{\mathcal{C}}$, which is $(\mathcal{D}_\epsilon)$.

The next Lemma provides a bound for the initial solution x^0.

Lemma 15. *For each $p \in P$, $\lambda^*_{\mathcal{C}} \leq \Lambda(p) \leq 2Mp^T f(x^0)$. Furthermore, $f_m(x^0) \geq \frac{1}{2M}\lambda^*_{\mathcal{C}}$ for each $m = 1,\ldots,M$.*

Proof. The first inequality follows from duality. For the second inequality,

$$\begin{aligned}\Lambda(p) = \;& \max\{p^T f(x) \mid x \in B\} \;=\; \max\{\textstyle\sum_{m=1}^M p_m f_m(x) \mid x \in B\} \\ \leq\;& \textstyle\sum_{m=1}^M p_m \max\{f_m(x) \mid x \in B\},\end{aligned}$$

where $\max\{f_m(x) \mid x \in B\} = \Lambda(e_m)$. Since $\hat{x}^{(m)}$ is the solution computed by $ABS(e_m, 1/2)$, $f_m(\hat{x}^{(m)}) \geq \frac{1}{2}\Lambda(e_m)$ implies that $\Lambda(e_m) \leq 2f_m(\hat{x}^{(m)})$. Therefore, $\Lambda(p) \leq 2\sum_{m=1}^M p_m f_m(\hat{x}^{(m)})$. Using the concavity and non-negativity of f_m we get

$$f_m(\hat{x}^{(m)}) \;\leq\; \sum_{\ell=1}^M f_m(\hat{x}^{(\ell)}) \;\leq\; M f_m(1/M \sum_{\ell=1}^M \hat{x}^{(\ell)}) \;=\; M f_m(x^0).$$

Combining the two inequalities, we obtain

$$\Lambda(p) \;\leq\; 2\sum_{m=1}^M p_m f_m(\hat{x}^{(m)}) \;\leq\; 2M \sum_{m=1}^M p_m f_m(x^0) \;=\; 2Mp^T f(x^0).$$

Finally, $f_m(x^0) \geq \frac{1}{M} f_m(\hat{x}^{(m)}) \geq \frac{1}{M}\frac{1}{2}\Lambda(e_m) \geq \frac{1}{2M}\lambda^*_{\mathcal{C}}$.

Remember that the reduced potential value $\phi_t(f(x)) = \Phi_t(\theta(f(x)), f(x))$. The following two Lemmas are used to bound the number of iterations of the *while* loop in algorithm Improve.

Lemma 16. *For any two consecutive iterations of algorithm Improve, the vectors x and x' that it computes satisfy $\phi_t(f(x')) - \phi_t(f(x)) \geq tv(x,\hat{x})^2/(4M)$.*

Proof. Algorithm Improve sets $x' = (1-\tau)x + \tau\hat{x}$. Suppose that $\theta = \theta(f(x))$. Since f_m is concave and using the definition of $p_m(f(x))$,

$$\begin{aligned} f_m(x') - \theta &\geq (1-\tau)f_m(x) + \tau f_m(\hat{x}) - \theta \\ &= (f_m(x)-\theta)(1 + \tau\tfrac{f_m(\hat{x})-f_m(x)}{f_m(x)-\theta}) \\ &= (f_m(x)-\theta)(1 + \tfrac{\tau M}{t\theta}p_m(f(x))(f_m(\hat{x})-f_m(x))). \end{aligned}$$

Let $p = p(f(x))$ and $p_m = p_m(f(x))$. Using the definition of the step length τ and the inequality $v(x,\hat{x}) \leq 1$, we can bound the last expression by

$$\begin{aligned} &|\tfrac{\tau M}{t\theta}p_m(f_m(\hat{x}) - f_m(x))| \\ &\leq \tfrac{\tau M}{t\theta}p_m(f_m(\hat{x}) + f_m(x)) \\ &\leq \tfrac{\tau M}{t\theta}(p^T f(\hat{x}) + p^T f(x)) \\ &= \tfrac{v(x,\hat{x})}{2} \leq \tfrac{1}{2}. \end{aligned}$$

Since $\theta(f(x)) \leq \frac{\lambda_{\mathcal{C}}(f(x))}{1+t/M} < \lambda_{\mathcal{C}}(f(x)) = \min\{f_1(x), \ldots, f_M(x)\}$, we have $f_m(x) - \theta(f(x)) > 0$. Using the bound for the expression above, this implies $f_m(x') - \theta(f(x)) > 0$. Now we analyze the potential value $\phi_t(f(x'))$:

$$\phi_t(f(x')) = \Phi_t(\theta(f(x')), f(x')) \geq \Phi_t(\theta, f(x')) = \ln\theta + \frac{t}{M}\sum_{m=1}^{M}\ln(f_m(x') - \theta),$$

where $\theta = \theta(f(x))$. Using $\phi_t(f(x)) = \ln\theta + \frac{t}{M}\sum_{m=1}^{M}\ln(f_m(x) - \theta)$, we obtain

$$\phi_t(f(x')) \geq \phi_t(f(x)) + \frac{t}{M}\sum_{m=1}^{M}[\ln(f_m(x') - \theta) - \ln(f_m(x) - \theta)].$$

Using the bound for $f_m(x') - \theta$ and the property that the function $\ln(x)$ is monotone increasing, $\phi_t(f(x'))$ is at least

$$\begin{aligned} &\geq \phi_t(f(x)) + \tfrac{t}{M}\textstyle\sum_{m=1}^{M}[\ln[(f_m(x)-\theta)(1 + \tfrac{\tau M}{t\theta}p_m(f_m(\hat{x}) - f_m(x)))] - \ln(f_m(x)-\theta)] \\ &= \phi_t(f(x)) + \tfrac{t}{M}\textstyle\sum_{m=1}^{M}\ln[1 + \tfrac{\tau M}{t\theta}p_m(f_m(\hat{x}) - f_m(x))]. \end{aligned}$$

Now we use $\frac{\tau M}{t\theta} p_m(f_m(\hat{x}) - f_m(x)) \in [-1/2, 1/2]$ and $\ln(1+z) \geq z - z^2$ for $z \geq -1/2$, to get

$$\begin{aligned}
&\phi_t(f(x')) - \phi_t(f(x)) \\
&\geq \tfrac{t}{M} \textstyle\sum_{m=1}^{M} [\tfrac{\tau M}{t\theta} p_m(f_m(\hat{x}) - f_m(x)) - (\tfrac{\tau M}{t\theta})^2 p_m^2 (f_m(\hat{x}) - f_m(x))^2] \\
&= \tau \tfrac{p^T f(\hat{x}) - p^T f(x)}{\theta} - \tfrac{\tau^2 M}{t\theta^2} \textstyle\sum_{m=1}^{M} (p_m(f_m(\hat{x}) - f_m(x)))^2.
\end{aligned}$$

Furthermore $\sum_{m=1}^{M}(p_m f_m(\hat{x}) - p_m f_m(x))^2 \leq \sum_{m=1}^{M}(p_m f_m(\hat{x}) + p_m f_m(x))^2 \leq (\sum_{m=1}^{M}(p_m f_m(\hat{x}) + p_m f_m(x)))^2 = (p^T f(\hat{x}) + p^T f(x))^2$. Using the definition of $\tau = \frac{t\theta v(x,\hat{x})}{2M(p^T f(\hat{x}) + p^T f(x))}$ and $v(x,\hat{x}) = \frac{p^T f(\hat{x}) - p^T f(x)}{p^T f(\hat{x}) + p^T f(x)}$, we finally get

$$\begin{aligned}
\phi_t(f(x')) - \phi_t(f(x)) &\geq \tau \tfrac{p^T f(\hat{x}) - p^T f(x)}{\theta} - \tfrac{\tau^2 M}{t\theta^2}(p^T f(\hat{x}) + p^T f(x))^2 \\
&= \tfrac{t v(x,\hat{x})}{2M} \cdot \tfrac{p^T f(\hat{x}) - p^T f(x)}{p^T f(\hat{x}) + p^T f(x)} - \tfrac{t v(x,\hat{x})^2}{4M} \\
&= \tfrac{t v(x,\hat{x})^2}{2M} - \tfrac{t v(x,\hat{x})^2}{4M} = \tfrac{t v(x,\hat{x})^2}{4M}.
\end{aligned}$$

Lemma 17. *For any two points $x' \in B$ and $x \in B$ with $\lambda_{\mathcal{C}}(f(x)) > 0$ and $\lambda_{\mathcal{C}}(f(x')) > 0$,*

$$\phi_t(f(x')) - \phi_t(f(x)) \;\leq\; (1+t)\ln \frac{\Lambda(p)}{p^T f(x)},$$

where $p = p(f(x))$ is the price vector defined in (4).

Proof. Let $\theta = \theta(f(x))$, $\theta' = \theta(f(x'))$ and $\Lambda(p) = \max\{p^T f(x) | x \in B\}$. Using the definition of $p_m(f(x))$ and the concavity of $\ln(x)$, we get:

$$\begin{aligned}
\phi_t(f(x')) - \phi_t(f(x)) &= \ln \tfrac{\theta'}{\theta} + \tfrac{t}{M} \textstyle\sum_{m=1}^{M} \ln(\tfrac{f_m(x') - \theta'}{f_m(x) - \theta}) \\
&= \ln \tfrac{\theta'}{\theta} + \tfrac{t}{M} \textstyle\sum_{m=1}^{M} \ln(\tfrac{M p_m(f(x))(f_m(x') - \theta')}{t\theta}) \\
&= \ln \tfrac{\theta'}{\theta} + t \ln \tfrac{M}{t\theta} + \tfrac{t}{M} \textstyle\sum_{m=1}^{M} \ln(p_m(f(x))(f_m(x') - \theta')) \\
&\leq \ln \tfrac{\theta'}{\theta} + t \ln \tfrac{M}{t\theta} + t \ln(\tfrac{1}{M} \textstyle\sum_{m=1}^{M} p_m(f(x))(f_m(x') - \theta')) \\
&= \ln \tfrac{\theta'}{\theta} + t \ln \tfrac{1}{t\theta} + t \ln(p(f(x))^T (f(x') - \theta' e))
\end{aligned}$$

where $e = (1, \ldots, 1)^T$. The last equality follows from $\ln(\frac{1}{M} \sum_{m=1}^{M} p_m(f(x)) (f_m(x') - \theta')) = \ln(p(f(x))^T (f(x') - \theta' e)) - \ln(M)$. Now we use $p(f(x))^T f(x') \leq \Lambda(p(f(x)))$ (by the definition of $\Lambda(p(f(x))) = \max_{y \in B} p(f(x))^T f(y)$) and $p(f(x))^T(\theta' e) = \sum_{m=1}^{M} p_m(f(x))\theta' = \theta'$. Using these (in-)equalities we have

$$\phi_t(f(x')) - \phi_t(f(x)) \leq \ln \frac{\theta'}{\theta} + t \ln \frac{1}{t\theta} + t \ln(\Lambda(p(f(x))) - \theta').$$

The last expression is well defined, since $\Lambda(p(f(x))) - \theta' \geq p(f(x))^T f(x') - p(f(x))^T(\theta' e) = p(f(x))^T(f(x') - \theta e) > 0$. The last inequality follows from $\theta' < \lambda_{\mathcal{C}}(f(x')) \leq f_m(x')$ and $f(x') - \theta' e > 0$. To bound now the difference $\phi_t(f(x')) - \phi_t(f(x))$, we calculate the

$$\max_{y \in (0,\Lambda(p))} [\ln \frac{y}{\theta} + t \ln \frac{1}{t\theta} + t \ln(\Lambda(p) - y)]$$

where $p = p(f(x))$. This can be done by computing the first derivative of the function above. We get the maximum value for $y = \Lambda(p)/(1+t)$. This implies

$$\begin{aligned} \phi_t(f(x')) - \phi_t(f(x)) &\leq \ln \tfrac{\Lambda(p)}{(1+t)\theta} + t \ln \tfrac{1}{t\theta} + t \ln(\Lambda(p) \tfrac{t}{1+t}) \\ &= \ln \tfrac{\Lambda(p)}{(1+t)\theta} + t \ln \tfrac{\Lambda(p)}{(1+t)\theta} \\ &= (1+t) \ln \tfrac{\Lambda(p)}{(1+t)\theta} = (1+t) \ln \tfrac{\Lambda(p)}{p^T f(x)}. \end{aligned}$$

The last equality follows from Lemma 13 (c).

Now we are able to calculate the number of iterations of algorithm Improve.

Theorem 4. *Algorithm Improve solves* $(\mathcal{C}_\epsilon)$ *and* $(\mathcal{D}_\epsilon)$ *in* $O(M\epsilon^{-1}(\ln M + \epsilon^{-1}))$ *iterations.*

Proof. Let N_0 be the number of iterations to reach a point x^1 with corresponding error $v \leq 1/2$ starting from our initial solution x^0. For all iterations with $v \geq 1/2$, each iteration increases the potential by at least $tv^2/4M \geq t/16M$ (see Lemma 16). By Lemma 17, the total increase is bounded by $\phi_t(f(x^1)) - \phi_t(f(x^0)) \leq (1+t) \ln \frac{\Lambda(p(f(x^0)))}{p(f(x^0))^T f(x^0)}$. Since $t = \epsilon/6$ and $\Lambda(p(f(x^0))) \leq 2M\, p(f(x^0))^T\, f(x^0)$ (by Lemma 15), we obtain that

$$N_0 \leq \frac{(1+\epsilon/6)16M \ln(2M)}{\epsilon/6} = O(M\epsilon^{-1} \ln M).$$

Now suppose that the error $v_\ell = v(x^\ell, \hat{x}^\ell) \leq 1/2^\ell$ for an iteration with computed vector $x^\ell \in B$, and let N_ℓ be the number of iterations to halve this error (i.e. to obtain a vector $x^{\ell+1}$ with $v(x^{\ell+1}, \hat{x}^{\ell+1}) \leq v_\ell/2$). We get $\phi_t(f(x^{\ell+1})) - \phi_t(f(x^\ell)) \geq \frac{N_\ell t v_\ell^2}{16M}$. On the other hand, the definition of v_ℓ implies $p(f(x^\ell))^T f(\hat{x}^\ell)(1 - v_\ell) = p(f(x^\ell))^T f(x^\ell)(1 + v_\ell)$. Using $ABS(p(f(x^\ell)), t)$, we get a solution $\hat{x}^\ell$ with $p(f(x^\ell))^T f(\hat{x}^\ell) \geq (1-t)\Lambda(p(f(x^\ell)))$. Combining the two inequalities,

$$\frac{\Lambda(p(f(x^\ell)))}{p(f(x^\ell))^T f(x^\ell)} \leq \frac{(1+v_\ell)}{(1-t)(1-v_\ell)} \leq \frac{(1+v_\ell)}{(1-v_\ell)^2} \leq (1+10v_\ell).$$

The last two inequalities hold since $t \leq v_\ell \leq 1/2$. Lemma 17 implies that

$$\phi_t(f(x^{\ell+1})) - \phi_t(f(x^\ell)) \leq (1+t)\ln(1+10v_\ell) \leq 10(1+t)v_\ell.$$

This gives now an upper bound

$$N_\ell \;\le\; \frac{160M(1+t)v_\ell}{tv_\ell^2} \;=\; O(\frac{M}{\epsilon v_\ell}).$$

The total number of iterations can be obtained by summing N_ℓ over all $\ell = 0, 1, \ldots, \lceil \log(\frac{1}{t}) \rceil$. Therefore, the total number of iterations is bounded by

$$N_0 + O(M\epsilon^{-1} \sum_{\ell=1}^{\lceil \log(\frac{1}{\epsilon}) \rceil} 2^\ell) \;=\; O(M\epsilon^{-1}(\ln M + \epsilon^{-1})).$$

Next, we analyze the number of iterations of the scaling implementation with parameters $\epsilon_0 = (1 - 1/(2M))$ and $\epsilon_s = \epsilon_{s-1}/2$. The goal in the phase s is to compute a vector x^s with $\lambda_{\mathcal{C}}(f(x^s)) \ge (1-\epsilon_s)\lambda^*_{\mathcal{C}}$.

Theorem 5. *For any accuracy $\epsilon > 0$, the scaling implementation computes solutions x and p of $(\mathcal{C}_\epsilon)$ and $(\mathcal{D}_\epsilon)$, respectively, in*

$$N \;=\; O(M(\ln M + \epsilon^{-2}))$$

iterations.

Proof. To reach the first $\epsilon_0 \in (1/2, 1)$ in the primal and dual problem we need $O(M \ln M)$ iterations (by Theorem 4). Let N_s be the number of iterations in phase s to reach ϵ_s for $s \ge 1$. By Lemma 16, each iteration of phase s increases the potential function by at least $t_s^3/(4M) = \Theta(\epsilon_s^3/M)$. Lemma 17 implies that for $x = x^s$ and $x' = x^{s+1}$,

$$\phi_{t_s}(f(x^{s+1})) - \phi_{t_s}(f(x^s)) \;\le\; (1+t_s)\ln \frac{\Lambda(p(f(x^s)))}{p(f(x^s))^T f(x^s)}.$$

Note that x^s is a $\epsilon_{s-1} = 2\epsilon_s$ solution of $(\mathcal{C}_{\epsilon_{s-1}})$, and therefore $f(x^s) \ge (1 - 2\epsilon_s)\lambda^*_{\mathcal{C}} e$. Furthermore, the inequalities $\Lambda(p(f(x^s))) \le (1+2\epsilon_s)\lambda^*_{\mathcal{C}}$, $\Lambda(p(f(x^s))) \le \frac{1+2\epsilon_s}{1-2\epsilon_s}\lambda_{\mathcal{C}}(f(x^s)) \le \frac{1+2\epsilon_s}{1-2\epsilon_s} p(f(x^s))^T f(x^s)$ imply that

$$\frac{\Lambda(p(f(x^s)))}{p(f(x^s))^T f(x^s)} \le (1 + 8\epsilon_s).$$

Using $\ln(1+8\alpha) \le 8\alpha$ one can bound N_s by $O(M/\epsilon_s^2)$. Then, as before the overall number of iterations is bounded by

$$N_0 + \sum_{s \ge 1} N_s \;\le\; O(M \ln M) + O(M\epsilon^{-2}) \;=\; O(M(\ln M + \epsilon^{-2})).$$

Remark. The root $\theta(f(x))$ can be computed again only approximately. But an accuracy of $O(\epsilon^2/M)$ for $\theta(f(x))$ is sufficient to generate the above bounds on the number of iterations. With this required accuracy, the number of evaluations of the sum $\sum_{m=1}^{M} \frac{1}{f_m(x)-\theta}$ is bounded by $O(\ln(M\epsilon^{-1}))$. This gives $O(M\ln(M\epsilon^{-1}))$ arithmetic operations to determine $\theta(f(x))$ approximately. Again by using Newton's method the overhead can be improved to $O(M \ln\ln(M\epsilon^{-1}))$.

References

1. M. Aigner and G.M. Ziegler, Proofs from THE BOOK, Springer Verlag 1999.
2. A.K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, Scheduling independent multiprocessor tasks, *Algorithmica* 32 (2002), 247-261.
3. B. Baker, E. Coffman, and R. Rivest, Orthogonal packings in two dimensions, *SIAM Journal on Computing*, 9 (1980), 846-855.
4. D. Bienstock, Potential function methods for approximately solving linear programming problems: theory and practice, Kluwer, Boston, 2002.
5. J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, Scheduling in Computer and Manufacturing Systems, Springer Verlag, Berlin (1996).
6. B. Chen, C.N. Potts, and G.J. Woeginger, A review of machine scheduling: complexity, algorithms and approximability, in Handbook of Combinatorial Optimization, (D.-Z. Du and P.M. Pardalos, eds.), Kluwer, 1998, 21-169.
7. E. Coffman, M. Garey, D. Johnson, and R. Tarjan, Performance bounds for level-oriented two dimensional packing algorithms, *SIAM Journal on Computing*, 9 (1980), 808-826.
8. G.I. Chen and T.H. Lai, Scheduling independent jobs on hypercubes, *Proceedings 5th Symposium on Theoretical Aspects of Computer Science* (1988), LNCS 294, 273-280.
9. W.F. de la Vega and C.S. Lueker, Bin packing can be solved within $1+\epsilon$ in linear time, *Combinatorica*, 1 (1981), 349-355.
10. M. Drozdowski, Scheduling multiprocessor tasks - an overview, *European Journal on Operations Research*, 94 (1996), 215-230.
11. J. Du and J. Leung, Complexity of scheduling parallel task systems, *SIAM Journal on Discrete Mathematics*, 2 (1989), 473-487.
12. A.V. Fishkin, K. Jansen, and M. Mastrolilli, Grouping techniques for scheduling problems: simpler and faster, *European Symposium on Algorithms* ESA 2001, LNCS 2161, 2001, 206-218.
13. M.R. Garey and R.L. Graham, Bounds for multiprocessor scheduling with resource constraints, *SIAM Journal on Computing*, 4 (1975), 187-200.
14. N. Garg and J. Könemann, Faster and simpler algorithms for multicommodity flow and other fractional packing problems, *Proceedings 38th Symposium on Foundations of Computer Science* FOCS 98, 1998, 300-309.
15. M.D. Grigoriadis and L.G. Khachiyan, Coordination complexity of parallel price-directive decomposition, *Mathematics of Operations Research*, 21 (1996), pp. 321–340.
16. M.D. Grigoriadis, L.G. Khachiyan, L. Porkolab, and J. Villavicencio, Approximate max-min resource sharing for structured concave optimization, *SIAM Journal on Optimization*, 41 (2001), 1081-1091.
17. L.Hall, Approximation algorithms for scheduling, in: *Approximation Algorithms for NP-hard problems*, D.S. Hochbaum (ed.), PWS Publishing Company, Boston, 1995.
18. E. Horowitz and S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, *Journal of the Association for Computing Machinery* 23 (1976), 317-327.
19. K. Jansen and L. Porkolab, Improved approximation schemes for scheduling unrelated parallel machines, *Mathematics of Operations Research.* 26, 2001, 324-338.
20. K. Jansen and L. Porkolab, Linear-time approximation schemes for scheduling malleable parallel tasks, *Algorithmica* 32, 2002, 507-520.

21. K. Jansen and L. Porkolab, On preemptive resource constrained scheduling: polynomial-time approximation schemes, *Proceedings 9th International Conference on Integer Programming and Combinatorial Optimization*, IPCO 2002, LNCS 2337, 329-349.
22. K. Jansen, and H. Zhang, Approximation algorithms for general packing problems with modified logarithmic potential function, *Proceedings 2nd IFIP International Conference on Theoretical Computer Science*, TCS 2002, Foundations of information technology in the era of network and mobile computing, Kluwer Publisher, 255-266.
23. K. Jansen, Approximation algorithms for the general max-min resource sharing problem: faster and simpler, *Proceedings 9th Scandinavian Workshop on Algorithm Theory*, SWAT 2004, Humlebæk, LNCS 3111, 2004, 311-322.
24. N. Karmarkar and R.M. Karp, An efficient approximation scheme for the one-dimensional bin-packing problem, *Proceedings 23rd IEEE Symposium on Foundations of Computer Science*, FOCS 82, 1982, 312-320.
25. C. Kenyon and E. Remila, Approximate strip packing, *Mathematics of Operations Research* 25 (2000), 645-656.
26. B. Korte and J. Vygen, Combinatorial Optimization: Theory and Algorithms, *Algorithms and Combinatorics* 21, Springer Verlag, 2000.
27. E. Lawler, Fast approximation algorithms for knapsack problems, *Mathematics of Operations Research*, 4 (1979), 339-356.
28. J.K. Lenstra, D.B. Shmoys, and E. Tardos, Approximation algorithms for scheduling unrelated parallel machines, *Mathematical Programming*, 24 (1990), 259-272.
29. W. Ludwig and P. Tiwari, Scheduling malleable and nonmalleable parallel tasks, *Proceedings 5th ACM-SIAM Symposium on Discrete Algorithms* (1994), 167-176.
30. S.A. Plotkin, D.B. Shmoys, and E. Tardos, Fast approximation algorithms for fractional packing and covering problems, *Mathematics of Operations Research*, 20 (1995), 257-301.
31. I. Schiermeyer, Reverse-Fit: A 2-optimal algorithm for packing rectangles, *Proceedings 2nd European Symposium of Algorithms* (1994), LNCS 855, 290-299.
32. A. Steinberg, A strip-packing algorithm with absolute performance bound two, *SIAM Journal on Computing*, 26 (1997), 401-409.
33. J. Turek, J. Wolf, and P. Yu, Approximate algorithms for scheduling parallelizable tasks, *Proceedings 4th ACM Symposium on Parallel Algorithms and Architectures* (1992), 323-332.
34. J. Villavicencio and M.D. Grigoriadis, Approximate Lagrangian decomposition with a modified Karmarkar logarithmic potential, *Network Optimization*, P. Pardalos, D.W. Hearn and W.W. Hager (eds.), Lecture Notes in Economics and Mathematical Systems 450 (1997), 471-485.
35. J. Villavicencio and M.D. Grigoriadis, Approximate structured optimization by cyclic block coordinate descent, *Applied Mathematics and Parallel Computing*, H. Fisher et al. (eds.), Physica Verlag (1996), 359-371.
36. N.E. Young, Randomized rounding without solving the linear program, *Proceedings 6th ACM-SIAM Symposium on Discrete Algorithms* (1995), 170-178.
37. Y. Zhu and M. Ahuja, On job scheduling on a hypercube, *IEEE Transactions on Parallel Distributed Systems*, 4 (1993), 62–69.

A Simpler Proof of Preemptive Total Flow Time Approximation on Parallel Machines[⋆]

Stefano Leonardi

Dipartimento di Informatica e Sistemistica,
University of Rome "La Sapienza"
leon@dis.uniroma1.it

Abstract. We consider the classical problem of scheduling jobs in a multiprocessor setting in order to minimize the flow time (total time in the system). The performance of the algorithm, both in offline and online settings, can be significantly improved if we allow preemption: i.e., interrupt a job and later continue its execution. Preemption is inherent to make a scheduling algorithm efficient [7,8]. Minimizing the total flow time on parallel machines with preemption is known to be NP-hard on $m \geq 2$ machines. Leonardi and Raz [8] showed that the well known heuristic *shortest remaining processing time* (SRPT) performs within a logarithmic factor of the optimal offline algorithm on parallel machines. It is not known if better approximation factors can be reached and thus SRPT, although it is an online algorithm, becomes the best known algorithm in the off-line setting. In fact, in the on-line setting, Leonardi and Raz showed that no algorithm can achieve a better bound.

In this work we present a nicer and simpler proof of the approximation ratio of SRPT. The proof presented in this paper combines techniques from the original paper of Leonardi and Raz [8] with those presented in a later paper on approximating total flow time when job preemption but not job migration is allowed [2] and on approximating total flow time non-clairvoyantly [3], that is when the processing time of a job is only known at time of completion.

1 Introduction

One of the most basic performance measures in multiprocessor scheduling problems is the overall time the jobs are spending in the system. This includes the delay of waiting for service as well as the actual service time. This measure captures the overall quality of service of the system. We consider the classical problem of minimizing the total flow time in a multiprocessor setting with jobs released over time. More formally, we consider a set of n jobs and m identical parallel machines. Every job j has processing time p_j and release time r_j. The

[⋆] Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), IST-2001-32007 (APPOL2), and by the MURST Projects "Algorithms for Large Data Sets: Science and Engineering" and "Resource Allocation in Computer Networks".

flow time of a job is the time interval it spends in the system between release and completion. The total flow time of the set of jobs is the sum of the individual flow times of the n jobs.

The performance of the algorithm, both in offline and online settings, can be significantly improved if we allow preemption: i.e., interrupt a job and later continue its execution, perhaps migrating it to a different machine. As shown below, preemption is inherent to make a scheduling algorithm efficient. In the non-preemptive case it is impossible to achieve a "reasonable" approximation. Specifically, even for one machine one cannot achieve an approximation factor of $O(n^{\frac{1}{2}-\epsilon})$ unless $NP = P$ [7]. For $m > 1$ identical parallel machines it is impossible to achieve an approximation factor of $O(n^{\frac{1}{3}-\epsilon})$ unless $NP = P$ [8]. Thus, preemptions really seem to be essential.

Minimizing the flow time on one machine with preemption can be done optimally in polynomial time using the natural algorithm shortest remaining processing time (SRPT) [4]. For more than one machine the preemptive problem becomes NP-hard [5]. Only very recently, Leonardi and Raz [8] showed that SRPT achieves logarithmic approximation for the multiprocessor case, showing a tight bound of $O(\log(\min\{n/m, P\}))$ on $m > 1$ machines with n jobs, where P denotes the ratio between the processing time of the longest and the shortest jobs. In the offline setting, it is not known if better approximation factors can be reached in polynomial time. In fact, in the on-line setting SRPT is optimal, i.e., no algorithm can achieve a better bound up to a constant factor[8]. For the easier problem of minimizing the total completion time a constant approximation and even a PTAS can be obtained [6,1]

The analysis of SRPT we report in this paper is still based on the ideas from the original work of Leonardi and Raz [8]. In a later paper Awerbuch, Azar, Leonardi and Regev [2] presented an algorithm that achieves an $O(\log P)$ and an $O(\log n)$ approximation factor for the problem of minimizing total flow time with preemption when job migration is not allowed, i.e. preempted jobs must be resumed later on the same machine they were run at the time of preemption. The analysis of the algorithm proposed in [2] borrows several ideas from [8] however classifyng jobs into classes allows to remove several difficulties from the analysis. The proof of the $O(\log P)$ approximation for SRPT follows the lines of the proof in [2].

In a recent paper Becchetti and Leonardi [3] consider non-clairvoyant scheduling algorithms to minimize the total flow time. In the non-clairvoyant scheduling problem the existence of a job, but not its processing time, is known at time of release. The processing time of the job is only known when the job is actually completed. In [3], a randomized variation of the Multi-level-feedback algorithm, widely used for processor scheduling in time sharing operating system such as Unix and Windows NT, is proved to achieve a tight $O(\log n)$ competitive ratio for a single machine system and an $O(\log n \log \frac{n}{m})$ competitive ratio for a parallel machine system. A remarkable simplification of the proof of the $O(\log n)$ approximation factor for SRPT draws ideas from the techniques introduced in [3] and from a technical Lemma presented in [2].

2 The Model

We are given a set J of n jobs and a set of m identical machines. Each job j is assigned a pair (r_j, p_j) where r_j is the release time of the job and p_j is its processing time. In the preemptive model a job that is running can be preempted and continued later on any machine. The scheduling algorithm decides which of the jobs should be executed at each time. Clearly a machine can process at most one job at any given time and a job cannot be processed before its release time. For a given schedule define C_j to be the completion time of job j in this schedule. The flow time of job j for this schedule is $F_j = C_j - r_j$. The total flow time is $\sum_{j \in J} F_j$. The goal of the scheduling algorithm is to minimize the total flow time for each given instance of the problem. In the off-line version of the problem all the jobs are known in advance. In the on-line version of the problem each job is introduced at its release time and the algorithm bases its decision only upon the jobs that were already released.

Shortest Remaining Processing Time (SRPT) schedules at any time those jobs with shortest remaining processing time for a maximum number of m. It follows that a job preemption happens only when a newly released job has processing time shorter than the remaining processing time of a job on execution. When a job is completed, that job with shortest remaining processing time currently not assigned to any machine, if any, is scheduled. This results in at most n preemptions operated by SRPT along the execution of the algorithm.

For a given input instance J and a scheduling algorithm S, we denote by $F^S(J)$ the total flow time of the schedule computed by S on input J. Denote by $F^{OPT}(J)$ the minimum value of the total flow time on input istance J. A schedule S is c-approximate if for every input instance J, $F^S(J) \leq cF^{OPT}(J)$. In the following we will omit J when clear from the context.

3 Analysis of SRPT

We denote by A the scheduling algorithm $SRPT$ and by OPT the optimal off-line algorithm that minimizes the flow time for any given instance. Whenever we talk about time t we mean the moment after the events of time t happened. A job is called *alive* at time t for a given schedule if it has already been released but has not been completed yet. Our algorithm classifies the jobs that are alive into classes according to their remaining processing times. A job j whose remaining processing time is in $[2^k, 2^{k+1})$ is in *class* k for $-\infty < k < \infty$. Notice that a job changes its class during its execution. We denote the class of a job upon arrival as the *initial class* of a job.

For a given scheduling algorithm S we define $V^S(t)$ to be the volume of a schedule at a certain time t. This volume is the sum of all the remaining processing times of jobs that are alive. In addition, we define $\delta^S(t)$ to be the number of jobs that are alive. $\Delta V(t)$ is defined to be the volume difference between our algorithm and the optimal algorithm, i.e., $V^A(t) - V^{OPT}(t)$. We also define by $\Delta\delta(t) = \delta^A(t) - \delta^{OPT}(t)$ the alive jobs difference at time t between

A and OPT. For a generic function f (V, ΔV, δ or $\Delta\delta$) we use $f_{\geq h, \leq k}(t)$ to denote the value of f at time t when restricted to jobs of classes between h and k. Similarly, the notation $f_{=k}(t)$ will represent the value of function f at time t when restricted to jobs of class precisely k.

Let $\gamma^S(t)$ be the number of non-idle machines at time t. We denote by $\mathcal{T}$ the set of times in which $\gamma^A(t) = m$, that is, the set of times in which none of the machines is idle. We indicate with $\mathcal{S}$ also the size $\int_{t\in\mathcal{S}} dt$ of a set of times $\mathcal{S}$. Denote by P_{min} the processing time of the shortest job and by P_{max} the processing time of the longest job and $P = P_{max}/P_{min}$. Denote by $k_{min} = \lfloor \log P_{min} \rfloor$ and $k_{max} = \lfloor \log P_{max} \rfloor$ the classes of the shortest and longest jobs upon their arrival, that is the maximum and the minimum initial class of a job.

We start by observing the simple fact that the flow time is the integral over time of the number of jobs that are alive (for example, see [8]):

Fact 1. *For any scheduler S,*

$$F^S = \int_t \delta^S(t)dt.$$

The following is an obvious lower bound on the flow time of any schedule:

Lemma 1. $F^S \geq \sum_j p_j$.

Lemma 2. *There are at most $2 + \log P$ initial classes for a job.*

Proof. The number of initial classes of a job is at most $k_{max} - k_{min} + 1 \leq 2 + \log P$.

The proof of the following lemma is straightforward since the total time spent by the m machines processing jobs is excatly $\sum_j p_j$.

Lemma 3. $\int_t \gamma^A(t)dt = \sum_j p_j$.

Now, assume that $t \in \mathcal{T}$ and let $\hat{t} < t$ be the earliest time for which $[\hat{t}, t) \subset \mathcal{T}$. We denote the last time in which a job of class more than k was processed by t_k. In case such jobs were not processed at all in the time interval $[\hat{t}, t)$ we set $t_k = \hat{t}$. So, $\hat{t} \leq t_{k_{max}} \leq t_{k_{max-1}} \leq ... \leq t_{k_{min}} \leq t$.

Lemma 4. *For $t \in \mathcal{T}$, $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(t_k)$.*

Proof. Notice that in the time interval $[t_k, t)$, algorithm A is constantly processing on all the machines jobs whose class is at most k. The off-line algorithm may process jobs of higher classes. Moreover, that can cause jobs of class more than k to actually lower their classes to k and below therefore adding even more to $V^{OPT}_{\leq k}(t)$. Finally, the release of jobs of class $\leq k$ in the interval $[t_k, t)$ is not affecting $\Delta V_{\leq k}(t)$. Therefore, the difference in volume between the two algorithms cannot increase between t_k and t.

Lemma 5. *For $t \in \mathcal{T}$, $\Delta V_{\le k}(t_k) \le m2^{k+1}$.*

Proof. First we claim that at any moment $t_k - \epsilon$, for any $\epsilon > 0$ small enough, the algorithm holds $m_1 < m$ jobs whose class is at most k, with total processing time bounded by $m_1 2^{k+1}$. In case $t_k = \hat{t}$, at any moment just before t_k there is at least one idle machine. Otherwise, $t_k > \hat{t}$ and by definition we know that a job of class more than k is processed just before t_k. At time $t_k - \epsilon$, $m_2 \le m - m_1$ jobs of class $k+1$ may change their class to k. Moreover, at time t_k jobs of class at most k might arrive. However, these jobs increase both $V^{OPT}_{\le k}(t_k)$ and $V^{A}_{\le k}(t_k)$ by the same amount, so jobs that arrive exactly at t_k do not change $\Delta V_{\le k}(t_k)$ and can be ignored. Altogether, we have $\Delta V_{\le k}(t_k) \le (m_1 + m_2)2^{k+1} \le m2^{k+1}$.

Lemma 6. *For $t \in \mathcal{T}$, $\Delta V_{\le k}(t) \le m2^{k+1}$.*

Proof. Combining Lemma 4 and 5, we obtain $\Delta V_{\le k}(t) \le \Delta V_{\le k}(t_k) \le m2^{k+1}$

The claim of the following lemma states a property that will be used in the proof of both the $O(\log P)$ and the $O(\log \frac{n}{m})$ approximation results.

Lemma 7. *For $t \in \mathcal{T}$, for $k_{min} \le k_1 \le k_2 \le k_{max}$, $\delta^{A}_{\ge k_1, \le k_2}(t) \le m(k_2 - k_1 + 2) + 2\delta^{OPT}_{\le k_2}(t)$.*

Proof. $\delta^{A}_{\ge k_1, \le k_2}(t)$ can be expressed as:

$$
\begin{aligned}
\sum_{i=k_1}^{k_2} \delta^{A}_{=i}(t) &\le \sum_{i=k_1}^{k_2} \frac{\Delta V_{=i}(t) + V^{OPT}_{=i}(t)}{2^i} \\
&= \sum_{i=k_1}^{k_2} \frac{\Delta V_{\le i}(t) - \Delta V_{\le i-1}(t)}{2^i} + \sum_{i=k_1}^{k_2} \frac{V^{OPT}_{=i}(t)}{2^i} \\
&\le \frac{\Delta V_{\le k_2}(t)}{2^{k_2}} + \sum_{i=k_1}^{k_2-1} \frac{\Delta V_{\le i}(t)}{2^{i+1}} - \frac{\Delta V_{\le k_1-1}(t)}{2^{k_1}} + 2\delta^{OPT}_{\ge k_1, \le k_2}(t) \\
&\le 2m + \sum_{i=k_1}^{k_2-1} m + \delta^{OPT}_{\le k_1-1}(t) + 2\delta^{OPT}_{\ge k_1, \le k_2}(t) \\
&\le m(k_2 - k_1 + 2) + 2\delta^{OPT}_{\le k_2}(t).
\end{aligned}
$$

The first inequality follows since 2^i is the minimum processing time of a job of class i. The third inequality follows since the processing time of a job of class i is less than 2^{i+1}. The fourth inequality is derived by applying Lemma 6, observing that $\Delta V_{\le k_1-1}(t) \ge -V^{OPT}_{\le k_1-1}(t)$ and that 2^{k_1} is the maximum processing time of a job of class at most $k_1 - 1$. The claim of the lemma then follows.

The following corollary of Lemma 7 is used in the proof of the $O(\log P)$ approximation ratio of Theorem 2

Corollary 1. *For $t \in \mathcal{T}$, $\delta^A(t) \le m(4 + \log P) + 2\delta^{OPT}(t)$.*

Proof. We write

$$\begin{aligned} \delta^A(t) &= \delta^A_{\le k_{max}, \ge k_{min}}(t) + \delta^A_{<k_{min}}(t) \\ &\le m(k_{max} - k_{min} + 2) + 2\delta^{OPT}(t) + m \\ &\le m(4 + \log P) + 2\delta^{OPT}(t). \end{aligned}$$

The first inequality follows from the claim of Lemma 7 when $k_2 = k_{max}$ and $k_1 = k_{min}$, and from the fact that since a job of class less than k_{min} is never preempted, there are at any time t at most m jobs of class less than k_{min} in the SRPT schedule. The second inequality is obtained since $k_{max} - k_{min} \le \log P + 1$.

Theorem 2. *$F^A \le (6+\log P)\cdot F^{OPT}$, that is, algorithm SRPT has a $(6+\log P)$ approximation factor.*

Proof.

$$\begin{aligned} F^A &= \int_t \delta^A(t)dt \\ &= \int_{t \notin \mathcal{T}} \delta^A(t)dt + \int_{t \in \mathcal{T}} \delta^A(t)dt \\ &\le \int_{t \notin \mathcal{T}} \gamma^A(t)dt + \int_{t \in \mathcal{T}} ((4 + \log P)\gamma^A(t) + 2\delta^{OPT}(t))dt \\ &\le (4 + \log P) \int_t \gamma^A(t)dt + 2\int_{t \in \mathcal{T}} \delta^{OPT}(t)dt \\ &\le (4 + \log P) \sum_j p_j + 2\int_t \delta^{OPT}(t)dt \\ &\le (6 + \log P)F^{OPT} \end{aligned}$$

The first equality derives from the definition of F^A. The second is obtained by looking at the time in which none of the machines is idle and the time in which at least one machine is idle separately. The third inequality uses Corollary 1. The fifth inequality uses Lemma 3, while the sixth inequality follows from Lemma 1.

We now turn to prove the $O(\log \frac{n}{m})$ approximation ratio for $SRPT$. Let $\overline{k}$ be the maximum integer such that for some time $t \in \mathcal{T}$ $\delta^A_{\ge \overline{k}}(t) \ge m$. If not such integer exists, fix $\overline{k} = k_{min} - 1$. Let $\mathcal{T}_j \subseteq \mathcal{T}$, $j = k_{min} + 1, .., \overline{k}$, be the set of time instants when all machines are busy, at least one machine is busy with jobs of class j and no machine is busy with jobs of class higher than j. Let $\mathcal{T}_{\overline{k}+1} \subseteq \mathcal{T}$ be the set of time instants when all machines are busy and at least one machine is processing a job of class higher than $\overline{k}$.

Finally, let $\mathcal{T}_{k_{min}} \subseteq \mathcal{T}$ be the set of time intants when all machines are busy with jobs of class less than or equal to k_{min}. Observe that $\{\mathcal{T}_{k_{min}}, \ldots, \mathcal{T}_{\overline{k}+1}\}$ defines a partition of $\mathcal{T}$. We can write the total flow time of SRPT as:

$$\begin{aligned}
F^A &= \int_{t\notin\mathcal{T}} \delta^A(t)dt + \int_{t\in\mathcal{T}} \delta^A(t)dt \\
&= \int_{t\notin\mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} \int_{t\in\mathcal{T}_j} \delta^A(t)dt + \int_{t\in\mathcal{T}_{\overline{k}+1}} \delta^A(t)dt \\
&\leq \int_{t\notin\mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} \int_{t\in\mathcal{T}_j} (2m + \delta^A_{\geq j, \leq \overline{k}}(t))dt + \int_{t\in\mathcal{T}_{\overline{k}+1}} 2mdt \\
&\leq \int_{t\notin\mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} \int_{t\in\mathcal{T}_j} (4m + m(\overline{k}-j) + 2\delta^{OPT}_{\leq\overline{k}}(t))dt \qquad (1) \\
&\quad +2\int_{t\in\mathcal{T}_{\overline{k}+1}} mdt \\
&\leq \int_{t\notin\mathcal{T}} \gamma^A(t)dt + 4\int_{t\in\mathcal{T}} \gamma^A(t)dt + \sum_{j=k_{min}}^{\overline{k}} m(\overline{k}-j)\mathcal{T}_j + 2\int_{t\in\mathcal{T}} \delta^{OPT}(t)dt \\
&\leq 6F^{OPT} + \sum_{j=k_{min}}^{\overline{k}} m(\overline{k}-j)\mathcal{T}_j \qquad (2)
\end{aligned}$$

where the $\mathcal{T}_|$ in the last two lines denotes the total amount of time that is spent in $calT_j$. The third inequality follows since at any time $t \in \mathcal{T}_j, j = k_{min}, .., \overline{k}+1$, by definition of $\mathcal{T}_j$, there are at most m alive jobs of class less than j and, by definition of $\overline{k}$, at most m alive jobs of class bigger than $\overline{k}$ in the SRPT schedule. The fourth inequality derives by the application of Lemma 7. Finally, the fifth and the sixth inequalities use Lemma 3 and Lemma 1.

We are left to bound the term $F(n) = \sum_{j=k_{min}}^{\overline{k}} m(\overline{k}-j)\mathcal{T}_j$. We show this in the following Lemma:

Lemma 8.

$$F(n) = \sum_{j=k_{min}}^{\overline{k}} m(\overline{k}-j)T_j = O(\log\frac{n}{m})F^{OPT}.$$

Proof. We define $\mathcal{T}^l_j$ for $j \leq k_{min}+1$ to be the set of time instants in $\{t \geq 0\}$, thus also considering time instants with some machine idle, when machine l is processing a job of class j. $\mathcal{T}^{\updownarrow}_{\|\Updownarrow\rangle\backslash}$ denotes the time in which machine l is processing a job of class k_{min} or less. We first observe that:

$$F(n) = \sum_{j=k_{min}}^{\overline{k}} m(\overline{k}-j)\mathcal{T}_j$$

$$\leq \sum_{j=k_{min}}^{\overline{k}} \sum_{l=1}^{m} (\overline{k}-j)\mathcal{T}_j^l, \tag{3}$$

since every time $t \in \mathcal{T}_j$ is also part of m sets $\mathcal{T}_i^l$ with $i \leq j$.

Let $n_j, j = k_{min}, ..., \overline{k}-1$, be the number of jobs of initial class j in the input instance. Let $n_{\overline{k}}$ be the number of jobs of initial class bigger or equal than $\overline{k}$ released in the input instance. For sake of simplicity, we will refer in the following to jobs of initial class higher than $\overline{k}$ as jobs of initial class $\overline{k}$.

Now, observe that a job of initial class j gives a contribution to equation (3) bounded by $2\sum_{i=0}^{j-k_{min}}(\overline{k}-j+i)2^{j-i}$ since each job of initial class j has been processed for at most 2^i time units when in class $i = k_{min}+1, \ldots, j$. This contribution is, by simple algebraic manipulation, at most equal to $2(\overline{k}-j+1)2^{j+1}$.

We then continue with the following inequalities:

$$\begin{aligned} F(n) &\leq 2\sum_{j=k_{min}}^{\overline{k}} n_j(\overline{k}-j+1)2^{j+1} \\ &= 4\sum_{j=k_{min}}^{\overline{k}} n_j(\overline{k}-j)2^j + 2\sum_{j=k_{min}}^{\overline{k}} n_j 2^{j+1} \\ &\leq 4\sum_{j=k_{min}}^{\overline{k}} n_j 2^j(\overline{k}-j) + 4\sum_j p_j, \end{aligned} \tag{4}$$

since a job of initial class j has processing time at least equal to 2^j. We exchange variable number j with $i = \overline{k}-j$. Let $I_i = n_{\overline{k}-i}2^{\overline{k}-i}, i = 0, \ldots, \overline{k}-k_{min}$. The first term $\sum_{j=k_{min}}^{\overline{k}} n_j 2^j(\overline{k}-j)$ of equation (4) becomes:

$$\sum_{i=0}^{\overline{k}-k_{min}} iI_i. \tag{5}$$

We can derive an upper bound on $F(n)$ by maximizing function (5) subject to the two obvious constraints:

$$\sum_{i=0}^{\overline{k}-k_{min}} I_i \leq \sum_j p_j \tag{6}$$

$$\sum_{i=0}^{\overline{k}-k_{min}} \frac{I_i}{2^{\overline{k}-i}} \leq n. \tag{7}$$

Constraint (7) implies

$$\sum_{i=0}^{\overline{k}-k_{min}} I_i \leq n2^{\overline{k}}. \tag{8}$$

To complete the proof we need the following simple mathematical lemma proved in [2]:

Lemma 9. *Given a sequence $a_1, a_2, ...$ of non-negative numbers such that $\sum_{i\geq 1} a_i \leq A$ and $\sum_{i\geq 1} 2^i a_i \leq B$ then $\sum_{i\geq 1} ia_i \leq \log(4B/A)A$.*

Proof. Define a second sequence, $b_i = \sum_{j\geq i} a_j$ for $i \geq 1$. Then it is known that $A \geq b_1 \geq b_2 \geq ....$ Also, it is known that $\sum_{i\geq 1} 2^i a_i = \sum_{i\geq 1} 2^i(b_i - b_{i+1}) = \frac{1}{2}\sum_{i\geq 1} 2^i b_i + b_1$. This implies that $\sum_{i\geq 1} 2^i b_i \leq 2B$.

The sum we are trying to upper bound is $\sum_{i\geq 1} b_i$. This can be viewed as an optimization problem where we try to maximize $\sum_{i\geq 1} b_i$ subject to $\sum_{i\geq 1} 2^i b_i \leq 2B$ and $b_i \leq A$ for $i \geq 1$. This corresponds to the maximization of a continuous function in a compact domain and any feasible point where $b_i < A, b_{i+1} > 0$ is dominated by the point we get by replacing b_i, b_{i+1} with $b_i + 2\epsilon, b_{i+1} - \epsilon$. Therefore, it is upper bounded by assigning $b_i = A$ for $1 \leq i \leq k$ and $b_i = 0$ for $i > k$ where k is large enough such that $\sum_{i\geq 1} 2^i b_i \geq 2B$. A choice of $k = \lceil \log(2B/A) \rceil$ is adequate and the sum is upper bounded by kA from which the result follows.

We apply Lemma 9 to our problem with variables $a_i = I_i$, $i = 0, \ldots, \overline{k} - k_{min}$, $A = \sum_j p_j$ by constraint (6), $B = n2^{\overline{k}}$ by constraint (8), to obtain:

$$\begin{aligned}\sum_{i=0}^{\overline{k}-k_{min}} iI_i &\leq \log(\frac{4n2^{\overline{k}}}{\sum_j p_j}) \sum_j p_j \\ &\leq O(\log \frac{n}{m}) F^{OPT}, \end{aligned} \tag{9}$$

where the last inequality follows since, by the definition of $\overline{k}$, at some time t there are at least m jobs of class bigger or equal than $\overline{k}$, for which $\sum_j p_j \geq m2^{\overline{k}}$.

Combining equations (4) and (9) and from $\sum_j p_j \leq F^{OPT}$, we obtain the desired bound.

Finally, Lemma 8 together with equation (2) leads to our result:

Theorem 3. *Algorithm SRPT has an $O(\log \frac{n}{m})$ approximation factor.*

4 Conclusions

In this paper we present a simpler proof of the approximation of SRPT for preemptive minimization of the total flow time on parallel identical machines. The proof relies on the original ideas of [8] and on new tools of analysis introduced in later works [2,3]. A major open problem is to devise a constant approximation algorithm or even an approximation scheme for the problem.

References

1. F. N. Afrati, E. Bampis, C. Chekuri, D. R. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein and M. Sviridenko. Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates. Proceedings of the *40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pp. 32-44, 1999.
2. B. Awerbuch, Y. Azar, S. Leonardi and O. Regev. Minimizing the flow time without migration. *Proc. of the 31st annual ACM Symposim on Theory of Computing*, pp. 198-205, 1999.
3. L. Becchetti and S. Leonardi. Non-Clairvoyant scheduling to minimize the average flow time on single and parallel machines. *Proc. of the 33rd annual ACM Symposim on Theory of Computing*, pp. 94-103, 2001.
4. K.R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.
5. J. Du, J. Y. T. Leung, and G. H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75(3):347–355, 1990.
6. L. Hall, Andreas S. Schulz, D. Shmoys and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. In *Mathematics of Operations Research* 22, pp. 513-544, 1997.
7. H. Kellerer, T. Tautenhahn and G.J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM Journal on Computing*, Vol. 28, Number 4, pp. 1155-1166, 1999.
8. S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 110–119, El Paso, Texas, 1997. To appear in *Journal of Computer and System Sciences – special issue for STOC '97.*

Approximating a Class of Classification Problems

Ioannis Milis[*]

Department of Informatics,
Athens University of Economics and Business,
Patission 76, 10434 Athens, Greece
milis@aueb.gr

Abstract. We consider a general classification problem, also known as *labeling problem*, which is strongly related to several standard classification frameworks and has applications in various computer science domains. In this chapter, we put together and review known results coming from application domains as well as recent advances on the approximability of the problem.

1 Introduction

In several contexts we aim to classify a set of pairwise related objects by assigning to each of them one of a given set of labels (or classes), taking into account information we have about:

(i) the relative likelihood of assigning a label to an object,
(ii) the strength of pairwise relationships between objects, and
(iii) the similarity between labels.

The quality of such a classification is based on the trade-off between two competing facts: each object tends to be labeled with the most likely label, while strongly related objects tend to be labeled with similar labels. An objective function that naturally captures this trade-off is one based on the contribution of two kinds of cost: the cost of labeling each object individually and the cost of labeling each pair of related objects.

This classification problem and its variants arise in various contexts and are encountered in the literature under several names, with objects and labels corresponding to context related entities. The context independent statement of the problem we are using in this chapter was first introduced by Kleinberg and Tardos [58], who also called this the *labeling problem*.

Formally, the **Labeling Problem (LaP)** asks for assigning one label, from a set L of k labels, to each object in a set V of n pairwise related objects. The relative likelihood of assigning a label to an object is given in terms of an assignment cost matrix $c(u, a)$, $u \in V$, $a \in L$, with non-negative entries. The

[*] This work has been partially supported by European Commission, Project APPOL (IST-1999-14084).

pairwise relationships between objects are represented by a weighted undirected graph $G = (V, E)$, where an edge $(u, v) \in E$ indicates that objects u and v are related and a positive weight $w(u, v)$ represents the strength of the their relationship. In addition, we have a distance function $d(\cdot, \cdot)$ on the set L of labels, representing the (un)similarity between labels.

A labeling of V over L is a function $f : V \rightarrow L$ and the quality of such a labeling is based on the contribution of two terms of cost:

(i) The *assignment cost*, $A(V)$, for labeling individually all objects. Labeling an object $u \in V$ with $f(u)$ contributes a cost $c(u, f(u))$ to $A(V)$.
(ii) The *separation cost*, $S(E)$, for labeling all pairs of related objects. Labeling two related objects $u, v \in V$, $(u, v) \in E$, with $f(u)$ and $f(v)$, respectively, contributes a cost $w(u, v) \cdot d(f(u), f(v))$ to $S(E)$.

Thus, the LaP asks for a labeling f that minimizes the total cost given by

$$C(f) = A(V) + S(E) = \sum_{u \in V} c(u, f(u)) + \sum_{(u,v) \in E} w(u, v) \cdot d(f(u), f(v)).$$

1.1 LaP in Various Contexts

The LaP model has been used, under different names, to formulate classification problems arising mainly in two computer science domains: distributed software engineering and image processing. The study of LaP in these contexts gave some interesting results and established its close relation with well known classification frameworks, including the classical quadratic assignment problem and well known graph partition problems in combinatorial optimization, and the theory of Markov Random Fields in statistics. In this subsection we present the connections of LaP to these frameworks and its applications.

Historically, LaP was first introduced by Stone [71] in the context of distributed software engineering, as a *task assignment problem*. In this context, objects correspond to the tasks of a modular application and labels correspond to the processors of a distributed system. Because of different processors' speeds and/or resources, the execution time of tasks varies from one processor to another and the assignment cost matrix $c(u, a), u \in V, a \in L$, represent these execution times. The graph $G = (V, E)$ is known as the task graph. Edges $(u, v) \in E$ represent data communication between tasks and weights $w(u, v)$, $(u, v) \in E$, indicate the amount of data to be exchanged between tasks u and v. A data exchange between tasks incurs a communication overhead due to protocols and transmission delays. Systems' communication links are, in general, heterogeneous and distances $d(a, b), a, b \in L$ represent the overhead incurred for transferring a unit of data between processors a and b. Thus, the separation cost, $w(u, v) \cdot d(a, b)$, for a pair of tasks u and v assigned to processors a and b, respectively, corresponds to the communication overhead incurred. In this context we aim to assign tasks to processors in order to minimize the sum of the execution time of all tasks plus the communication overhead incurred by all pairs of communicating tasks. The objective function $C(f)$ represents exactly this quantity which can be interpreted as the total time the application occupies the system resources (both processors and communication links) [71].

The reader may have already noticed a relation between LaP and the *Quadratic Assignment Problem* (QAP), one of the most difficult and extensively studied combinatorial optimization problems (see for example [13] for a recent thorough survey). In the context of QAP, the nodes of the graph $G = (V, E)$ traditionally represent $|V|$ machines and the weights $w(u, v)$, $(u, v) \in E$, represent flows of some material between machines. The set L represents exactly $|L| = |V|$ locations, and $d(a, b)$ is the distance between locations a and b. The aim of QAP is to assign each machine to a *distinct* location in order to minimize an objective function consisting, also, of two types of cost: an operating cost $c(u, a)$ for assigning machine u at location a, and a flow cost $w(u, v) \cdot d(a, b)$ incurred if machines u and v are assigned to locations a and b respectively. In other words, QAP asks for the minimization of $C(f)$ over all *bijections*, while LaP asks for the minimization of $C(f)$ over all *functions* f. Clearly, the *uncapacitated QAP*, in which more than one machines can be assigned to the same location is equivalent to the LaP variant where $|V| = |L|$. Note, that both the ucapacitated QAP and LaP become trivial in the absence of the assignment cost term, while the QAP is NP-hard whether such a term is included or not. Magirou and Milis [66] studied the relation between LaP and QAP and used a polynomially solvable special case of LaP to obtain Lagrangean relaxation lower bounds for QAP.

The LaP is also closely related to the theory of *Markov Random Fields (MRFs)* [20,56] in statistics. In this context, assume that we are given a labeling $f'(u)$ for each object $u \in V$ obtained from a *true* labeling f by a perturbation of independent random noise at each object. The question is to decide the most probable true labeling from the given labeling f' and the underlying MRF. A MRF is defined by a graph on the set of objects, whose edges represent dependencies between objects, and the assumption made that the conditional probability of labeling an object by a given label depends only on the labeling of its neighbors. A standard approach to decide the most probable true labeling is to find the labeling f that maximizes the *a posteriori* probability $\Pr[f'|f]$ which by Bayes' Law is proportional to $\Pr[f'|f] \cdot \Pr\ [f]$. Kleinberg and Tardos [58] observed that this last problem is equivalent to LaP, if the underlying MRF satisfies two properties, which, however, match with most applications: (i) pairwise interactions among objects and (ii) spatial homogeneity of the MRF. MRFs have been used as a core analytical framework in several domains including image processing [43], biometric analysis [6], language modeling [29], hypertext documents categorization [16] and learning [31].

Geman and Geman [43] introduced the theory of MRFs in the field of image processing. A direct application of this model is the *image restoration problem*, where the goal is to restore an image corrupted by noise. We are given the grid of image pixels, which correspond to objects, and for each pixel we are also given an observed intensity that is possibly corrupted by noise. The labels correspond to discretized intensities and the goal is to assign a new intensity to each pixel in order to restore the real image. The quality of such a restoration is also based on the trade-off between two competing facts: each pixel tends to be labeled with an intensity close to its original value, while neighboring pixels tend to be labeled with similar intensities due to smoothness of real images (except in boundary regions of sharp discontinuity). Besides the image restoration problem, the MRFs model has been extensively used to model many other problems in the image processing domain [25,30,61], including the *visual correspondence problem* [68] which is the basis of determining the depth and the motion in an image. The LaP applies to these problems through its equivalence, under certain assumptions, to MRFs.

Problem variants: Several variants of the labeling problem, with respect to the distance function d on the set L of labels, are encountered in the literature motivated by both their practical and theoretical interest. In the general problem d can be any function on L, while other problem variants can be obtained by restricting d.

We say that (L, d) is a *metric* iff for every $p, q, r \in L$, $d(p, p) = 0$, $d(p, q) \geq 0$, $d(p, q) = d(q, p)$ and $d(p, q) + d(q, r) \geq d(p, r)$. Moreover, we say that (L, d) is a *uniform metric* if $d(a, b) = 1$, if $a \neq b$, and zero otherwise. In the *linear metric* case we consider w.l.o.g. that labels are consecutive integers, i.e. $L = \{1, 2, ..., k\}$, and the distance function is defined as $d(a, b) = |a - b|$. If the set of labels contains non-consecutive numbers, i.e. $L \subset [1, k]$, then we can consider an infinite assignment cost for labeling an object by a label $a \notin L$. The *truncated linear metric* function is defined as $d(a, b) = min\{M, |a - b|\}$ and it is the most natural (non-uniform) robust linear metric. It is clear that for $M = 1$ the truncated linear metric reduces to the uniform metric and that for $M > k$ it reduces to the liner metric.

Non-metric distance functions are also of practical interest, especially in the image processing context. A *convex linear* function is defined as $d(a, b) = g(|a - b|)$, where g is convex and increasing. In this case, d is a non-metric function, since (L, d) is a metric iff g is concave and increasing. The *quadratic linear* function $d(a, b) = |a - b|^2$ is such a convex linear one. On the other hand, the *truncated quadratic linear* function $d(a, b) = min\{M, |a - b|^2\}$ is a non-convex, non-metric linear function.

In Figure 1, we present the relations between the variants of LaP as well as their connections to other known frameworks. The multiterminal cut problem is a generalization of the famous minimum-cut/maximum flow problem. This problem as well as its generalization known as 0-extension problem, are two well-studied graph partition problems related to LaP as it is shown in Figure 1. Because of the central role that these two problems play in the study of LaP they are discussed in details in Section 2. Note, that most of the known results for LaP are based on its relation to the multiterminal cut problem.

1.2 Approximation Algorithms

A *ρ-approximation algorithm* computes polynomially a solution within a factor ρ of the optimum one. In this setting we are interested in designing ρ-approximation algorithms with ρ as small as possible as well as on finding lower bounds on ρ. When ρ is a constant, then we say that the approximation algorithm is a *constant factor* one. We say, however, that we have a *Polynomial Time*

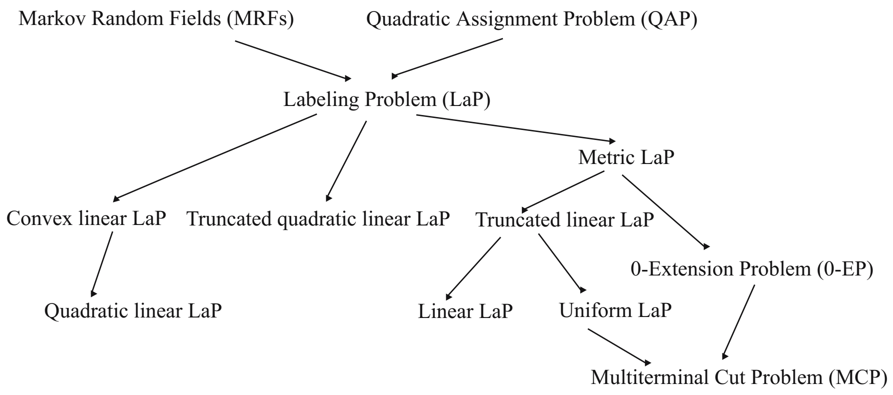
Markov Random Fields (MRFs)
Quadratic Assignment Problem (QAP)
Labeling Problem (LaP)
Metric LaP
Convex linear LaP
Truncated quadratic linear LaP
Truncated linear LaP
0-Extension Problem (0-EP)
Quadratic linear LaP
Linear LaP
Uniform LaP
Multiterminal Cut Problem (MCP)

Fig. 1. Relations and variants of LaP (arrows are oriented from general to special problems

Approximation Scheme (PTAS) if we give an algorithm which, for any fixed value of ϵ, can construct an $(1+\epsilon)$-approximation solution. If the time complexity of a PTAS is also polynomial in $1/\epsilon$, then it is called *Fully Polynomial Time Approximation Scheme* (FPTAS). For negative approximation results, the notion of NP-hardness is used to prove lower bounds on approximation factors or to disprove the existence of a FPTAS, unless P=NP. It is well known that, there is no FPATS for *strongly* NP-hard problems and that there is no PTAS for MAX SNP-hard problems, unless P=NP.

A general and successful technique of designing approximation algorithms is based on linear programming (LP) relaxations of a problem. First, an optimum fractional solution to such an LP-relaxation is polynomially obtained. This solution, whose cost is denoted by $\bar{C}$, is then used to obtain an approximate solution to the original problem. One method to this end is to round the fractional solution to an integral feasible one of cost $C \leq \rho\bar{C}$ (another method is based on the primal-dual schema). Randomization is often helpful to this rounding, giving a solution of expected cost $\mathbf{E}[C_R] \leq \rho\bar{C}$. Such a rounding procedure sometimes can be derandomized obtaining a solution of cost $C \leq \mathbf{E}[C_R]$. Obviously, C (resp. $\mathbf{E}[C_R]$) is a deterministic (resp. randomized) ρ-approximate solution, since $\bar{C}$ is lower bound of the cost C^* of an optimum integral solution, i.e. $\bar{C} \leq C^* \leq C \leq \mathbf{E}[C_R]$.

However, there is a gap between the cost of the optimum (integral) solution and the cost of a fractional optimum one. This gap is formally defined as the ratio $\frac{C^*}{\bar{C}}$ and it is known as *integrality gap* of the LP-relaxation. Clearly, any ρ-approximate solution achieved this way inherits this gap and, therefore, it is quite interesting to determine a lower bound, γ, on the integrality gap, i.e. a number $\gamma \leq \frac{C^*}{\bar{C}} \leq \rho$. If $\gamma = \rho$, then we say that the integrality gap of an LP-relaxation is ρ. For a deep and elegant presentation of LP-based approximation algorithms and techniques the reader is referred to [74].

1.3 Organization of the Chapter

In the next section we establish the relation between the multiterminal cut problem, the 0-extension problem and the LaP, and review approximation results for these two graph partition problems. The techniques used in this context have been later generalized to offer optimal or approximate solutions for several LaP variants. In Section 3, we review early results for LaP coming from the distributed software engineering community. In Section 4 we present local search approximation algorithms, originated in the image processing context. In Section 5 we review recent LP-based approximation algorithms. Throughout Sections 3-5, the material of this chapter is classified according the problem's variants defined above. We conclude in Section 6 with a summary of known results and open questions.

2 Related Graph Partition Problems

The famous minimum-cut/maximum flow problem is one of the simplest graph partition problems. Recall that given a graph $G = (V, E)$ with positive edge weights $w(e), e \in E$ and two terminal nodes $s, t \in V$, the problem asks for the minimum weight set of edges Φ such that the removal of Φ from E disconnects s from t. The most direct generalization of this classical problem is known as **multiterminal or multiway cut problem (MCP)**[28], and it is obtained by fixing a set $T \subset V$ of k terminal nodes:

Find a set of edges $\Phi \subseteq E$ such that
(i) the removal of Φ from E disconnects each terminal node from the others, and
(ii) $C(\Phi) = \sum_{e\in\Phi} w(e)$ is minimized.

Obviously, the removal of the edges in Φ from G creates exactly k connected components, each one containing a terminal node. In other words, Φ implies naturally an assignment of the non-terminal nodes $V \setminus T$ to the k terminal nodes T. Thus, MCP can be alternatively rewritten as following:

Find an assignment $f : V \rightarrow T$, such that
(i) $f(t) = t$, for all $t \in T$, and
(ii) $\sum_{(u,v)\in E} w(u, v) \cdot d(f(u), f(v))$ is minimized, where (T, d) is the uniform metric.

Recall that a metric (T, d) is called uniform if for every $a, b \in L$, $d(a, b) = 1$, if $a \neq b$, and zero otherwise. A natural generalization of MCP can be obtained by allowing (T, d) to be an arbitrary metric, instead of being the uniform metric. This generalization is known as **0-extension problem (0-EP)**, due to its alternative statement proposed by Karzanov [54] (see Section 2.2).

Both MCP and 0-EP can be related to LaP by considering that the non-terminal nodes correspond to objects and the terminal nodes correspond to labels. The statements of these problems differ, however, in two points: (i) The labels of LaP are distinct from the nodes of G, while the terminals nodes in MCP and 0-EP are a subset of the nodes of G, and (ii) The non-terminal nodes in MCP and 0-EP do not have any labeling preferences like assignment costs in LaP. Despite these facts, it is easy to see that MCP and 0-EP are special cases of LaP: Consider a MCP or 0-EP instance on a graph $G = (V, E)$ with positive edge weights $w(e), e \in E$, a set $T \subset V$ of k terminal nodes and (T, d) being the uniform metric for MCP or an arbitrary metric for 0-EP. Then, a LaP instance with objects set $V \setminus T$ and labels set T can be easily constructed by consider assignment costs

$$c(u,t) = \sum_{t' \in T \setminus \{t\}} w(u,t') \cdot d(t,t'), \quad u \in V \setminus T, \quad t \in T.$$

The cost of a solution to this LaP instance is:

$$\sum_{u \in V \setminus T} c(u, f(u)) + \sum_{\substack{(u,v) \in E \\ u,v \in V \setminus T}} w(u,v) \cdot d(f(u), f(v)) = \sum_{\substack{(u,v) \in E \\ \{u,v\} \not\subset T}} w(u,v) \cdot d(f(u), f(v)).$$

That is, the cost of a solution to the original MCP or 0-EP instance is equal, up to an invariant cost due to edges between terminal nodes, to the cost of a LaP instance.

2.1 Complexity and Approximability Results for the MCP

The study of MCP goes back to a 1983 unpublished but widely circulated preliminary version of [28] by Dahlhaus et al. They shown that MCP for arbitrary graphs is NP-hard (even for $k = 3$ and all weights equal to 1) by a reduction from SIMPLE MAX CUT. For the case of planar graphs, they gave a polynomial algorithm, for fixed k, and proved that MCP becomes NP-hard if k is part of the instance (even for all weights equal to 1), by a reduction from PLANAR THREE SATISFIABILITY. Moreover, they have shown that MCP is Max SNP-hard, for any fixed $k \geq 3$, and, hence, there is no a PTAS for MCP, unless P=NP. Hence, the NP-hardness and the Max SNP-hardness of both 0-EP and LaP, on arbitrary graphs, follow from these results.

Chopra and Rao [23] proved that for trees and 2-trees the general MCP can be solved in linear time by a straightforward dynamic programming algorithm. In fact, this result can be generalized to graphs of bounded tree-width for any fixed bound. The facets of the multiterminal cut polyherdron are also studied in [22,23,26]. Erdös and Székely [32,33] considered the problem of extending a partial k-coloring of a graph which is equivalent to MCP for general graphs (see [28] for a discussion on the relation between the two problems). PTASs for MCP on dense unweighted graphs has been proposed by Arora et al. [2] and Frieze and Kannan [39].

We contrast here the MCP with other known graph partition problems of related interest, but incomparable to LaP. In the k-cut problem [45,48,49,53,69], we are also given G, w and k as in MCP (but not terminal nodes T) and the goal is to find the minimum weight set of edges, whose removal separates the nodes of the graph G into exactly k nonempty connected components. Garg et al. in [42] consider a variation of MCP on node-weighted graphs which asks for the minimum weight set of nodes such that their removal disconnects each terminal from the others as well as the directed variant of MCP. This last variant was studied also by Naor and Zosin [67]. Finally, the *multipair cut* problem [40,41,57,72], proposed by Hu [50] as an integral dual to maximum multicommodity flow, can be viewed as another generalization of the MCP. In this problem we are given a list of pairs of terminals and the goal is to disconnect each pair in the list. Note, that MCP is a special case of the multi-pair cut problem in which the nodes in the list of pairs we are given form a clique.

A $2-2/k$-approximation algorithm for the MCP

Dahlhaus et al. [28] presented the following simple combinatorial algorithm, and derived the first approximation result for the MCP.

Algorithm D+:

```
    for each terminal node $t \in T$ do
        merge all terminals, but $t$, into $\bar{t}$ ; (set $w(u,\bar{t}) = \sum_{t' \in T \setminus t} w(u,t')$ )
        find a minimum $t - \bar{t}$ cut ;
    output the union of the cheapest $k-1$ cuts ;
```

The removal of the union of the cheapest $k-1$ minimum $t-\bar{t}$ cuts from G disconnects each terminal from all others and therefore it forms a multiterminal cut. On the other hand, the removal of an optimal multiterminal cut from G disconnects, also, each terminal node from the others, but does not induce optimal $t-\bar{t}$ cuts for each terminal node individually. Let V_t (resp. V_t^*), $t \in T$, be the set of nodes of G that remain connected to the terminal t after the removal of a minimum $t-\bar{t}$ cut (resp. a minimum multiterminal cut).

Theorem 1. **[28]** *Algorithm D+ is a tight* $(2-2/k)$*-approximation algorithm for the MCP.*

Proof. Let C be the cost of the multiterminal cut returned by Algorithm D+, and C_t, $t \in T$ be the cost of a minimum $t-\bar{t}$ cut. Let also C^* be the cost of an optimal multiterminal cut and C_t^* be the cost of $t-\bar{t}$ cut it induces, i.e. the cost of edges having exactly one of their endpoints in V_t^*. We have $C_t \leq C_t^*$, since C_t is the cost of a minimum cut separating t from all other terminals, and $\sum_{t \in T} C_t^* = 2C^*$, since each edge is counted twice in this summation. Thus,

$$C = \sum_{t \in T} C_t - \max_{t \in T} C_t \leq (1-\frac{1}{k}) \sum_{t \in T} C_t \leq (1-\frac{1}{k}) \sum_{t \in T} C_t^* \leq (2-\frac{2}{k}) C^*$$

For the tightness of the $2-2/k$ approximation factor of Algorithm D+ consider the graph $G=(V,E)$ with vertices $V=\{u_1,u_2,...,u_k,t_1,t_2,...,t_k\}$, edges $E=\{(u_i,u_{i+1\ (mod\ k)}) | 1 \leq i \leq k\} \bigcup \{(u_i,t_i) | 1 \leq i \leq k\}$, and edge weights $w(e)=1$, for $(u_i, u_{i+1\ (mod\ k)})$ edges, and $w(e) = 2 - \epsilon$, $\epsilon > 0$, for (u_i, t_i) edges. For each terminal node t_i the minimum weight $t_i - \bar{t_i}$ cut is unique and consists of the edge (u_i, t_i) of weight $2 - \epsilon$. The weight of the multiterminal cut constructed by the algorithm is $(2-\epsilon)(k-1)$. On the other hand, the optimal cut consists of all the k (u_i, u_{i+1}) edges, each of weight 1. Thus the approximation factor for such an instance is $(2-\epsilon)(k-1)/k$ which matches asymptotically the $2-2/k$. Assuming that the algorithm always breaks ties in the worst possible way (in terms of the total weight of the cut constructed) and taking $\epsilon = 0$ the approximation factor for this instance matches precisely the $2 - 2/k$. ∎

It is proven in [28] that there exists an optimum multiterminal cut for G that, for all $t \in T$ leaves all nodes in V_t connected to t, i.e. $V_t \subseteq V - t^*, t \in T$. Therefore, the size of a MCP instance can be reduced by $\sum_{t \in T}(|V_t| - 1)$ by simply merging all nodes in each $V_t, t \in T$, into the terminal t, since a minimum multiterminal cut for this reduced instance induces an optimal multiterminal cut for the original one. In order to maximally reduce the size of the original instance it is enough, for all $t \in T$, to compute the minimum $t - \bar{t}$ cut such that the induced set V_t is of maximum cardinality [28]. Such a $t - \bar{t}$ cut is unique and the V_t set it induces contains the sets induced by all other minimum $t - \bar{t}$ cuts. Moreover, it can be found by a linear time processing of the output a $t - \bar{t}$ minimum cut computation [37]. Thus, k minimum cut computations suffice to obtain a maximally reduced instance that still induces an optimal multiterminal cut.

Magirou [63] presents a different idea for applying minimum $t - \bar{t}$ cuts. Instead of merging all terminals, except t, into $\bar{t}$, his method considers $\bar{t}$ to be, for each non-terminal node, the most competitive to t terminal node, i.e. the weights of edges $w(u, \bar{t})$ are set to $w(u, \bar{t}) = \min_{t' \in T \setminus \{t\}} w(u, t')$. Everything, discussed above for Algorithm D+ holds also for this idea. Moreover, if V_t' is the set of nodes of G that remain connected to the terminal t after the removal of the minimum $t - \bar{t}$ cut obtained this way, it is shown that $|V_t'| \geq |V_t|$. Computational results reported in [63] show a considerably improved reduction of the instance size for moderate sized problems.

For the special cases of $k = 4$ and $k = 8$, Alon (see [28]) proposed a modification of Algorithm D+, obtaining improved approximation factors. In fact, this modification leads to a 4/3-approximation factor (instead of 3/2) for $k = 4$ and 12/7 (instead of 7/4) for $k = 8$. Cunningham [26] reports that this improvement for $k = 4$ has been, also, obtained, independently, by Zang. Unfortunately, this modification does not yield approximation factors better than $2 - 2/k$ for any values of k other than 4 and 8.

LP-relaxations of the MCP were also proposed to beat the $2 - 2/k$ approximation factor, as the MCP can be formulated as an integer program in various ways. For example, the uniform metric statement of the problem leads directly to the following integer program, with variables $d(u, v)$, $u, v \in V$, [14]:

$$\begin{array}{lll} \text{(MIP)} : \text{Minimize} & \sum_{(u,v) \in E} w(u,v) \cdot d(u,v) & \\ \quad\text{subject to} & (V, d) \text{ is a metric} & (1) \\ & d(t,t') = 1, \quad \forall t, t' \in T, t \neq t' & (2) \\ & d(u,v) \in \{0,1\}, \quad \forall u, v \in V & (3) \end{array}$$

Any solution d to (MIP) corresponds to a multiterminal cut: if $d(u,v) = 0$, then the nodes u and v belong to the same component; otherwise they belong to different components. Constraints (2) guarantee that each one of the k terminal nodes belongs to a different component, and the triangle inequalities implied by constraints (1) guarantee that each non-terminal node belongs to exactly one of these k components. An LP-relaxation of MCP can be obtained by replacing the integer constraints (3) of (MIP) by $0 \leq d(u,v) \leq 1, u, v \in V$. However, the integrality gap of this LP-relaxation is at least $2 - 2/k$. To see this consider a k-leaf star, with all its leaves being terminals nodes and all edge weights equal to 1. The minimum multiterminal cut of this instance is of cost $k-1$ (all edges but one are in the cut) and the optimal fractional solution is of cost $k/2$ ($d(u,v) = 1/2, \ \forall u, v \in V$). Thus, an integrality gap of at least $2 - 2/k$ follows.

Bertsimas et al. [5] presented a nonlinear formulation of the MCP and proposed several polynomial time solvable LP-relaxations, as well as a simple randomized rounding technique, obtaining a $2 - 2/k$ approximation factor too.

A $3/2 - 1/k$-approximation algorithm for the MCP

The $2 - 2/k$ approximation factor for MCP remained the best known one for over fifteen years, until a substantial improvement was obtained by Calinesku at al. [14]. They proposed a novel metric LP-relaxation by transforming MCP to a geometric problem of mapping the nodes of G into the vertices of the k-simplex

$$\Delta_k = \{x \in \mathbb{R}^k | (x \geq 0) \text{ and } \sum_{i=1}^{k} x_i = 1\}.$$

Δ_k is a convex polytope in $\mathbb{R}^k$ with k vertices. The vertices of Δ_k are the points $e^t, \ 1 \leq t \leq k$, where e^t is the unit k-dimensional vector with $e^t_t = 1$ and all other coordinates 0. To express a multiterminal cut and its cost, every node $u \in V$ is associated with a k-dimensional vector x^u in Δ_k. Then, a multiterminal cut is viewed as a mapping of the nodes of the graph G into the vertices of Δ_k, such that:

(i) every terminal node $t \in T$ is mapped to the vertex e^t of Δ_k (i.e. $x^t = e^t$, for all $t \in T$), and
(ii) every non-terminal node $u \in V \setminus T$ is mapped to some vertex e^t of Δ_k (i.e. $x^u = e^t$, for some $t \in T$).

For the cost of such a mapping (cut) consider the distance between the end-points of an edge $(u,v) \in E$ and denote the L_1 norm of a vector x in $\mathbb{R}^k$ by $\| x \|$. Then, it is clear that $d(u,v) = \| x^u - x^v \| = 2$, if u and v are mapped to different vertices of Δ_k (i.e. (u,v) is in the cut), and $d(u,v) = 0$, otherwise. Thus, $d(u,v) = \frac{1}{2} \| x^u - x^v \| = \frac{1}{2} \sum_{t=1}^{k} |x^u_t - x^v_t|$.

By relaxing the constraint of mapping the non-terminal nodes of G into the vertices of Δ_k and allowing them to be mapped anywhere in the simplex, the following LP-relaxation of MCP is obtained.

$$\begin{array}{lll}
\text{(SLP)} : \text{Minimize} & \frac{1}{2}\sum_{(u,v)\in E} w(u,v)d(u,v) & \\
\text{subject to} & d(u,v) = \frac{1}{2}\sum_{t=1}^{k} |x_t^u - x_t^v|, \forall u,v \in V & (1)\\
& x^u \in \Delta_k, \qquad\qquad \forall u \in V & (2)\\
& x^t = e^t, \qquad\qquad \forall t \in T & (3)
\end{array}$$

It can be easily verified that (SLP) is linear: Constraints (3) are clearly linear. Constraints (2), by the definition of the Δ_k, mean $x_t^u \geq 0,\ \forall u \in V,\ \forall t \in T$, and $\sum_{t=1}^{k} x_t^u = 1\ \forall u \in V$. Finally, the absolute value $|x_t^u - x_t^v|$ in constraints (1) can be expressed by introducing variables $x_t^{uv}, \forall u,v \in V,\ \forall t \in T$, such that $x_t^{uv} \geq x_t^u - x_t^v$ and $x_t^{uv} \geq x_t^v - x_t^u$.

Rounding a fractional solution to (SLP), to an integer one corresponds to placing all nodes at simplex vertices and thus to a multiterminal cut. The next lemma, based on the additivity of L_1 norm, is the key for obtaining an amenable to analysis rounding procedure.

Lemma 1. [14] *A fractional solution x to (SLP) for a graph $G = (V,E)$ can be transformed, in $O(kn^2)$ time, into a* normalized fractional solution $\tilde{x}$ *for a graph $\tilde{G} = (\tilde{V},\tilde{E})$ such that:*

(i) $|\tilde{V}|$ is $O(kn^2)$ and a multiterminal cut of $\tilde{G}$ implies a multiterminal cut of G.
(ii) the cost of $\tilde{x}$ for $\tilde{G}$ is at most the cost of x for G, and
(iii) for all edges $(u,v) \in \tilde{E}$, the vectors $\tilde{x}^u$ and $\tilde{x}^v$ differ in at most two coordinates.

Given this lemma, let x to be the optimal normalized solution to (SLP). Let $E_t \subseteq E$ be the set of edges whose corresponding vectors x^u and x^v differ in coordinate t. Note, that each edge $(u,v) \in E_t$ appears and in another set $E_{t'}$, since in a normalized solution the vectors x^u and x^v either are equal or differ in exactly two coordinates. Let W_t be the contribution of the edges in E_t to the cost of the solution x, i.e. $W_t = \sum_{(u,v)\in E_t} w(u,v)d(u,v)$. Using this notation the rounding procedure given below provides a multiterminal cut in terms of the sets V_t (the nodes remaining connected to terminal $t \in T$ after the removal of a cut).

Rounding procedure:

```
rename the terminals such that $W_t,\ t \in T$ is maximized at $t = k$;
choose at random a permutation $\sigma = \langle\sigma(1), \sigma(2), ..., \sigma(k)\rangle$
    from $\{\langle 1,2,...,k-1,k\rangle, \langle k-1,k-2,...,2,1,k\rangle\}$;
choose a real $\theta$ uniformly at random from (0,1);
for $t = 1$ to $k-1$ do
    $B(t,\theta) = \{u \in V | x_t^u \geq \theta\}$; (the set of nodes that are close to terminal $t$)
    $V_{\sigma(t)} = B(t,\theta) - \bigcup_{j<t} V_{\sigma(j)}$;
$V_k = V - \bigcup_{j<k} V_j$;
```

Theorem 2. [14] *There is a deterministic polynomial* $(3/2 - 1/k)$*-approximation algorithm for the MCP.*

Proof. It is clear that the rounding procedure produces a feasible solution to MCP. Let Φ be the set of edges in the multiterminal cut. The proof is based to the following two facts, proved in [14]:

(i) If $(u,v) \in E - E_k$, then $\mathbf{Pr}[(u,v) \in \Phi] \leq \frac{3}{2}\, d(u,v)$, and
(ii) If $(u,v) \in E_k$, then $\mathbf{Pr}[(u,v) \in \Phi] \leq d(u,v)$.

Due to the linearity of expectation, the expected value of the integer solution C_R, obtained by the rounding procedure is:

$$\begin{aligned}
\mathbf{E}[C_R] &= \sum_{(u,v)\in E} w(u,v)\mathbf{Pr}[(u,v) \in \Phi] \\
&= \sum_{(u,v)\in E\setminus E_k} w(u,v)\mathbf{Pr}[(u,v) \in \Phi] + \sum_{(u,v)\in E_k} w(u,v)\mathbf{Pr}[(u,v) \in \Phi] \\
&= \frac{3}{2} \sum_{(u,v)\in E} w(u,v)d(u,v) - \frac{1}{2} \sum_{(u,v)\in E_k} w(u,v)d(u,v) = \frac{3}{2}\bar{C} - \frac{1}{2}W_k,
\end{aligned}$$

where $\bar{C}$ is the cost of the fractional solution. Moreover, $\sum_{t=1}^{k} W_t = 2\bar{C}$ (recall that every edge appeared into two sets $E_t,\ t \in T$) and since $W_t,\ t \in T$ is maximized at $t = k$, it satisfies $W_k \geq \frac{2}{k}\bar{C}$. Therefore,

$$\mathbf{E}[C_R] \leq \frac{3}{2}\bar{C} - \frac{1}{k}\bar{C} = (\frac{3}{2} - \frac{1}{k})\bar{C}.$$

The rounding procedure can be easily implemented in (deterministic) polynomial time, except the step of choosing uniformly θ from $(0,1)$. However, even this step can be derandomized as follows: There are two possible choices for σ. For a given choice (permutation) of σ, two different choices of $\theta,\ \ \theta_1 < \theta_2$ produce combinatorially distinct mutliterminal cuts only if there is a terminal t and a node u such that $\theta_1 \leq x_t^u < \theta_2$. These "interesting" values of θ are at most $k|\tilde{V}|$ (recall that the fractional solution is assumed normalized) and they can be determined by sorting the nodes according to each coordinate separately. The resulting discrete sample space for (σ, θ) is of size $2k|\tilde{V}|$, that is $O(n^2k^2)$, since, by Lemma 1, $|\tilde{V}|$ is $O(kn^2)$. Since there is a choice of $\sigma,\ \theta$ that gives an integer solution of value at most $(\frac{3}{2} - \frac{1}{k})\bar{C}$, it follows that this solution can be found by searching exhaustively this sample space, instead of choosing θ from a continuous distribution. ∎

After this seminal result of Călinescu et al. [14], there was also a significant progress on exploring the (SLP) relaxation in order to derive both better approximation factors (by improved rounding procedures) and lower bounds on its integrality gap (by exhibiting certain MCP instances). Note, that the k-leaf star instance, used above for the LP-relaxation of (MIP), yields an integrality gap of 1 for (SLP).

For $k = 3$, Călinescu et al. in [14] gave a MCP instance yielding a 16/15 inegrality gap for (SLP). Cunningham and Tang [27] and Karger et al. [52] proposed, independently, a rounding procedure tailored to the case of $k = 3$ and obtained a 12/11-approximation algorithm. Moreover, both groups proved that this factor matches the integrality gap of (SLP) for this case, by exhibiting a certain class of graphs. For $k > 3$, Karger et al. [52], proposed also improved rounding schemes. For $k = 4$ and $k = 5$ they shown approximation factors of 1.1539 and 1.2161 respectively, based on computer constructed and analysed rounding schemes yielding, while for $k > 6$, they gave a single algorithm obtaining an (analytic) approximation factor of $1.3438 - \epsilon_k,\ \epsilon_k > 0$ which beats the $3/2 - 1/k$ factor, for all $k > 6$. They also evaluated computationally the value of ϵ_k, proving, this way, that $1.3438 - \epsilon_k < 3/2 - 1/k$, for all k. On the other hand, Călinescu et al. in [14] gave also an instance of the MCP yielding integrality gap of 13/12 for $k = 4$. However, Freund and Karloff [38] gave a family of instances having a $8/(7 + \frac{1}{k-1})$ integrality gap for every $k \geq 3$. This lower bound matches with the 13/12 for $k = 4$, and it is the only known for $k \geq 5$.

The reader can be also referred to the book of V. V. Vazirani [70, Chapters 4 and 19] for an excellent presentation of approximation algorithms for MCP, related problems and open questions.

2.2 Approximation Results for the 0-EP

The 0-EP is a generalization of the MCP and a special case of the LaP, with its name coming from its alternative statement based on the next definition.

Definition 1. *A metric (V, δ) is an extension of the metric (T, d), $T \subset V$, iff $\delta(t, t') = d(t, t')$, for all $t, t' \in T$.*
If, in addition, for every $u \in V$ there exists a unique $t \in T$ such that $\delta(u, t) = 0$, then (V, δ) is a 0-extension of (T, d).

Recall that given a graph $G = (V, E)$ with positive edge weights $w(e), e \in E$ and a set $T \subset V$ of k terminal nodes, the 0-EP asks for an assignment $f : V \to T$, such that

(i) $f(t) = t$, for all $t \in T$, and
(ii) $\sum_{(u,v)\in E} w(u, v) \cdot d(f(u), f(v))$ is minimized, where (T, d) is an arbitrary metric.

By Definition 1, it is clear that a 0-extension (V, δ) of (T, d) corresponds to an assignment $f : V \to T$ such that $f(u) = t$ if $\delta(u, t) = 0$. Therefore, the 0-EP can be stated as following:

Find a 0-extension (V, δ) of (T, d) that minimizes $\sum_{(u,v)\in E} w(u, v) \cdot \delta(u, v)$.

Karzanov [54] proposed a relaxation of the above formulation, asking for the extension (instead of the 0-extension) of (T, d) to (V, δ) that minimizes $\sum_{(u,v)\in E} w(u, v) \cdot \delta(u, v)$. This relaxation is known as *metric* relaxation of the 0-EP:

$$\begin{array}{rll}\text{(MLP)}: \text{Minimize} & \sum_{(u,v)\in E} w(u,v)\delta(u,v) & \\ \text{subject to} & (V,\delta) \text{ is a metric} & (1) \\ & \delta(t,t') = d(t,t') \quad \forall t,t' \in T & (2)\end{array}$$

If an assignment $f^* : V \to T$ defines an optimal solution to the 0-EP, then putting $\delta(u,v) = d(f^*(u), f^*(v))$ defines a feasible solution to (MLP) of the same cost with the optimal solution. Therefore, the cost of an optimal solution to (MLP) is a lower bound on the cost of an optimal solution to 0-EP.

(MLP) relaxation is clearly linear, and was used by Karzanov [54,55] to show that certain special cases (special metrics) of the 0-EP can be solved optimally in polynomial time. Recently, Călinescu et al. obtained an $O(logk)$-approximation algorithm for 0-EP [15], using this LP-relaxation. In fact, they proposed the following procedure to round an optimal fractional solution, to (MLP) (i.e. an extension of (T,d) to (V,δ)) to an integral one (i.e. a 0-extension of (T,d) to (V,δ)), that is, an assignment $f : V \to T$.

Rounding procedure:

```
for $t = 1$ to $k$ do $f(t) = t$;
for all $u \in V$ do $B_u = \min_{t\in T} \delta(u,t)$;
choose a random permutation $\sigma = \langle\sigma(1), \sigma(2), ..., \sigma(k)\rangle$ of the terminal nodes;
choose a real $\theta$ uniformly at random from [1,2);
for $t = 1$ to $k$ do
    for all unassigned nonterminal nodes $u$ do
        if $\delta(u,\sigma(t)) \leq \theta B_u$ then $f(u) = \sigma(t)$;
```

Theorem 3. [15] *There is a randomized $O(\log k)$-approximation algorithm for the 0-EP.*

Proof. Let u be non-terminal node, $t \in T$ be a terminal node with $\delta(u,t) = B_u$, and choose j such that $t = \sigma(j)$. If u is not assigned to a terminal in iterations $1, 2, ..., j-1$, it will be assigned to t in iteration j, since $\theta \geq 1$. Therefore, the assignment f produced by the rounding procedure is a solution to 0-EP. The expected cost of this assignment f is

$$\mathbf{E}[C_R] = \sum_{(u,v)\in E} w(u,v)\mathbf{E}[d(f(u), f(v))].$$

Let $\bar{\delta}_(u,v),\ u,v \in V$, be the optimal solution to (MLP). It is proven in [15] that, for any pair of distinct nodes $u,v \in V$,

$$\mathbf{E}[d(f(u), f(v))] \leq 38\mathcal{H}_k\delta(u,v),$$

where $\mathcal{H}_k = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{k}$ is the k-th harmonic number. Therefore,

$$\mathbf{E}[C_R] \leq 38\mathcal{H}_k \sum_{(u,v)\in E} w(u,v)\delta(u,v) \leq O(\log k)\bar{C} \leq O(\log k)C^*,$$

where $\bar{C}$ is the cost of the cost of the optimal solution to (MLP).

Therefore, there is a choice of σ and θ that produces a solution of cost $O(\log k)\bar{C}$. For a given choice (permutation) of σ, two different choices of θ, $\theta_1 > \theta_2$ produce combinatorial distinct assignments (0-extensions) only if there is a terminal t and a node u such that $\theta_2 B_u \leq \theta_1 B_u$. These "interesting" values of θ are at most $k|V|$ and they can be determined by sorting the fractions of $\bar{\delta}(u,t)/B_u$ over all nodes $u \in V$ with $B_u > 0$ and over all $t \in T$. Therefore, a solution of cost $O(\log k)\bar{C}$ can be found in randomized (due to the choice of σ) polynomial time by searching for θ exhaustively, instead of choosing it from a continuous distribution. ∎

Moreover, Călinescu et al. [15], proposed an improved rounding procedure of the optimal solution to (MLP) tailored to the special case of planar graphs. This improved rounding procedure guarantees an $O(r^3)$ approximation factor for the case of $K_{r,r} - minor$ free graphs. This becomes a $O(1)$ approximation factor for the case of planar graphs, since they are $K_{3,3} - minor$ free graphs. Finally, they also proved a lower bound of $O(\sqrt{\log k})$ on the integrality gap of (MLP).

The results of Călinescu et al. [15], and especially their result for the planar case as well as their lower bound on the integrality gap of (MLP), can be viewed as a counterpart of results proposed in the context of the multipair cut problem [40,41,57,72]. Recall that this last problem can be viewed as another generalization of MCP, not directly comparable to 0-EP, in the sense that neither problem is a special case of the other. It is, however, interesting that techniques proposed in this context, such as region growing and expander graphs, can be adapted in the case of 0-EP.

3 Early Results for the LaP

The first results for LaP were obtained in the context of distributed software engineering, where LaP was stated as a task assignment problem. Throughout this section, this task assignment problem is referred as LaP, tasks are referred as objects and processors as labels. Although MCP has not been yet stated, Stone [71] established a close relationship MCP and the uniform metric LaP. This relationship motivated the study of MCP [28,14] and inspired a large body of work during the '80's. In this section we present the results based on this LaP-MCP relationship as well as additional results for LaP obtained in this context. Surprisingly, this LaP-MCP relationship and almost the same results presented also in the late '90's in the context of image processing.

3.1 The $k = 2$ Case

For the case of $k = 2$ labels (processors) Stone [71] proposed a simple reduction to the minimum cut problem by constructing a weighted assignment graph $G' = (V', E')$ as follows: Two nodes are added to the original object (task) graph $G = (V, E)$, corresponding to labels $L = \{a, b\}$. Edges are also added from each node $u \in V$ to both a and b. That is, $V' = V \cup L$ and $E' = E \cup \{(u,a) | u \in V, a \in L\}$. The weights of the edges of the assignment graph G' are:

$w'(u, a) = c(u, b), u \in V$,
$w'(u, b) = c(u, a), u \in V$, and
$w'(u, v) = w(u, v), (u, v) \in E$.

Any $a-b$ cut of G' is associated to a solution of the original problem such that the nodes remaining connected to a and b, after the edges in cut are removed, are assigned the corresponding labels. Given the weights in G', it is obvious that the cost of any $a-b$ cut is equal to the cost of the associated labeling. Therefore, an optimal labeling is associated to a minimum $a-b$ cut of G' i.e. LaP is polynomial solvable for $k = 2$ labels. Note that the same result was, also, obtained, later, by Besag [7] and Greig et al. [46] in the context of image processing.

3.2 The Uniform Metric Case

Stone [71] tried also to extend his idea to the case of $k > 2$ labels and established the relationship between the uniform metric LaP and MCP. Boykov et al. [10] considered also the connection between the uniform LaP and MCP in the context of image processing.

Similarly with the $k = 2$ case, Stone proposed the construction of a k-label assignment graph $G' = (V', E')$. Now, k nodes are added to the original object graph $G = (V, E)$, each one corresponding to a label in L. Edges are also added from each node $u \in V$ to each of the nodes $a \in L$. That is, $V' = V \cup L$ and $E' = E \cup \{(u, a) | u \in V, a \in L\}$. The weights of the edges of the assignment graph G' are:

$$w'(u, a) = \frac{\sum_{b \in L} c(u, b)}{n-1} - c(u, a), u \in V, a \in L, \text{ and}$$
$$w'(u, v) = w(u, v), (u, v) \in E.$$

It is easy to verify that the cost of any multiterminal cut of G', with the L nodes considered as terminal nodes, is equal to the cost of the labeling associated to this cut, i.e. the objects remaining connected to node $a \in L$, after the edges in the cut are removed, are labeled a. After the introduction of the MCP by Dalhlaus et. al [28], Magirou and Milis [64] have shown formally the polynomial equivalence of uniform metric LaP and MCP.

Stone [71] has been also proposed the reduction of the size of a LaP instance based on the same idea which used later by Dahlhaus et al. [28] in the Algorithm D+ for the MCP (see Section 2.1): He considered an $a - \bar{a}$ minimum cut of G', where $a \in L$ and $\bar{a}$ a node formed by lumping together all the remaining labels $b \in L, b \neq a$ (i.e. the weights of the edges $(u, \bar{a}), u \in V$ are $w'(u, \bar{a}) = \sum_{b \in L \setminus \{a\}} w'(u, b)$). In fact, he proved that objects labeled a by a minimum $a - \bar{a}$ cut, retain this label in the optimal labeling. Thus the application of this process for all labels, results in a partial labeling which is consistent with the optimal one.

One can think that labeling all unlabeled objects by the label a, for which the $a - \bar{a}$ cut has the greatest cost, leads to a (2-2/k)-approximation algorithm for the uniform LaP as the Algorithm D+ for the MCP. Unfortunately, it is easy to see that the LaP to MCP reduction is not approximation preserving. Algorithm D+ requires non-negative edge weights and this does not hold for the $w'(u,a)$ weights of G'. Making these weights positive is not a problem: since for each object $u \in V$, the number of edges $(u,a), a \in L$, appearing in any multiterminal cut is exactly $k-1$, this can be done by adding to all of them an invariant depending on assignment costs $c(u,a), u \in V, a \in L$. However, the $2-2/k$ factor obtained by applying Algorithm D+ after this transformation refers to a LaP instance different than the original one. In fact, the cost of the LaP solution obtained this way can be arbitrarily far from the optimum solution of the original instance. Note that the modified idea proposed by Magirou [63] (see Section 2.1) for MCP suffers, also, from the same drawback.

Although the reduction of size of a LaP instance can not lead to an approximation algorithm, it was the starting point of the most known heuristic algorithm for the uniform metric LaP, proposed by Lo [62]. In fact, this algorithm deals with the labeling of objects that remain unlabeled after an iterative application of Stone's procedure. However, computational results reported in [63] show that for large sized problems both Stone's and Magirou's reductions tend to become useless. Kopidakis et al. [59] and Milis [65] presented different departures from the basic Stone's idea and proposed new heuristics, which outperform the Lo's one. Fernandez de la Vega and Lamari [36] studied special cases of uniform LaP (complete object graph and bi-valued assignment costs or constant weights) and proposed several approximation results.

3.3 The Linear Metric Case

In the linear metric case we consider w.l.o.g. $L = \{1, 2, ..., k\}$, i.e. the labels are the integers $1, 2, ..., k$, and $d(a,b) = |a-b|$. For this case a polynomial time algorithm was first proposed by Lee et al. [60] in the context of distributed software engineering. Boykov et al. [10,75] as well as Ishikawa and Geiger [51] obtained also the same result in the context of image processing. We present here the first algorithm proposed for this case by Lee et al. [60]. We call it **Algorithm L+**.

From the object graph $G = (V, E)$, an assignment graph $G' = (V', E')$ is constructed. This is done by copying $k-1$ times the graph G and adding a source and a sink node. The a-th copy of G, $1 \leq a \leq k-1$ is denoted by $G_a = (V^a, E^a)$ and the a-th copy of node $u \in V$ is denoted by $u_a \in V^a$. The source and the sink nodes are denoted by u_0 and u_k, respectively, and for notational convenience it is assumed that the source node u_o (resp. and the sink node u_k) represents the 0-th (resp. the k-th copy) of all nodes in V, i.e. $v_0 \equiv u_0$ and $v_k \equiv u_k$, for all $v \in V$. Hence, $V' = \{u_0, u_k\} \cup (\cup_{a=1}^{k-1}) V^a$. Edges $H_u = \{(u_a, u_{a+1}) | 0 \leq a \leq k-1\}$, for all $u \in V$, are also added in G'. Hence, $E' = \mathcal{E} \bigcup \mathcal{H}$, where $\mathcal{E} = \bigcup_{a=1}^{k-1} E^a$ and $\mathcal{H} = \bigcup_{u \in V} H_u$. Figure 2 shows this construction for an edge $(u,v) \in E$.

Let $\Phi \subset E'$ be the set of edges in a $u_0 - u_k$ cut of graph G'. Such a $u_0 - u_k$ cut is called *feasible* if $|\Phi \bigcap H_u| = 1$, for all $u \in V$ (an example of a feasible $u_0 - u_k$ cut is shown in Figure 2 by the dotted line). It is clear that a labeling is associated to any feasible cut, such that $f(u) = a+1$, if $\Phi \bigcap H_u = (u_a, u_{a+1})$.

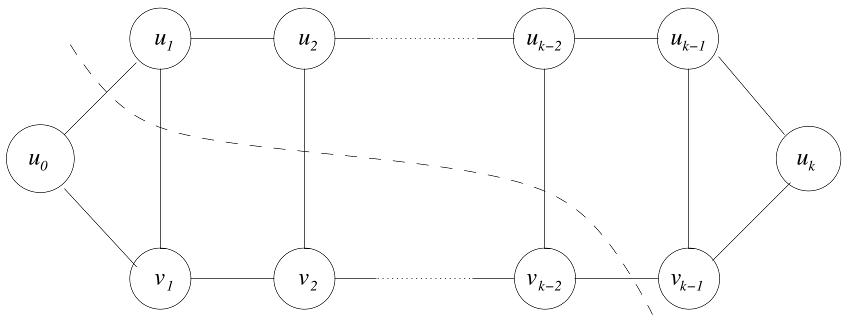

Fig. 2. The construction for the linear metric case

Theorem 4. [60] *If the weights of the edges of* G' *are:*
$w'(u_a, v_a) = w(u, v), u, v \in V, 1 \leq a \leq k-1,$ *and*
$w'(u_a, u_{a+1}) = c(u, a+1) + W, u \in V, 0 \leq a \leq k-1,$ *where* $W > (n-1)\sum_{e \in E} w(e),$
then an optimal labeling is associated to a minimum $u_0 - u_k$ *cut.*

Proof. Consider first the cost of a feasible $u_0 - u_k$ cut of the graph G'. The set Φ of the edges in such a cut can be written as $\Phi = \Phi_{\mathcal{H}} \bigcup \Phi_{\mathcal{E}}$, where $\Phi_{\mathcal{H}} = \Phi \bigcap \mathcal{H}$ and $\Phi_{\mathcal{E}} = \Phi \bigcap \mathcal{E}$. Let f be the labeling associated to this cut and $A(f)$ and $S(f)$ be its assignment and separation costs, respectively.

Since $u_0 - u_k$ is a feasible cut, it follows that:

(i) For each node $u \in V$, there exists an a, $0 \leq a \leq k-1$, such that $\Phi_{\mathcal{H}} \bigcap H_u = (u_a, u_{a+1})$, that is $w'(u_a, u_{a+1}) = c(u, a+1) + W = c(u, f(u)) + W$. Therefore, $C(\Phi_{\mathcal{H}}) = \sum_{u \in V}[c(u, f(u)) + nW] = nW + \sum_{u \in V} c(u, f(u)) = A(f) + nW$.
(ii) For each edge $(u, v) \in E$, there exist a, b, $0 \leq a, b \leq k-1$, such that $(u_a, u_{a+1}), (v_b, v_{b+1}) \in \Phi_{\mathcal{H}}$. Then, it is easy to see that exactly $|a-b|$ copies of the edge (u, v) (each of weight $w(u, v)$) belong to $\Phi_{\mathcal{E}}$. Since $|a-b| = d(f(u), f(v))$, it follows that $C(\Phi_{\mathcal{E}}) = \sum_{(u,v) \in E} w(u, v) d(f(u), f(v)) = S(f)$.

Hence, $C(\Phi) = C(\Phi_{\mathcal{H}}) + C(\Phi_{\mathcal{E}}) = A(f) + nW + S(f) = C(f) + nW$, and since nW is an invariant for all feasible cuts, a minimum feasible $u_0 - u_k$ cut is associated to an optimal labeling.

To complete the proof it is enough to prove that a minimum $u_0 - u_k$ cut is a feasible one. Let $\Phi^* \subset E'$ be a minimum $u_0 - u_k$ cut, and assume by contradiction that it is infeasible. It is $|\Phi^*_{\mathcal{H}} \bigcap H_u| \geq 1$, for all $u \in V$, for otherwise for some $u \in V$ the path $u_0, u_1, ..., u_k$ is not broken by Φ (recall that Φ is a $u_0 - u_k$ cut). Therefore, a feasible cut Φ can be constructed from Φ^* such that $|\Phi_{\mathcal{H}} \bigcap H_u| = 1$, for all $u \in V$. Clearly $\Phi_{\mathcal{H}}$ has at least one fewer $\mathcal{H}$-edge than Φ^*_H, and hence, $C(\Phi^*_{\mathcal{H}}) - C(\Phi_{\mathcal{H}}) > W$. Moreover,

$$C(\Phi_{\mathcal{E}}) = \sum_{(u,v) \in \mathcal{E}} w(u, v) d(f(u), f(v)) \leq \sum_{(u,v) \in \mathcal{E}} w(u, v)(n-1) < W.$$

Therefore,

$$\begin{aligned} C(\Phi^*) - C(\Phi) &= (C(\Phi^*_{\mathcal{E}}) + C(\Phi^*_{\mathcal{H}})) - (C(\Phi_{\mathcal{E}}) + C(\Phi_{\mathcal{H}})) \\ &= (C(\Phi^*_{\mathcal{H}}) - C(\Phi_{\mathcal{H}})) + (C(\Phi^*_{\mathcal{E}}) - C(\Phi_{\mathcal{E}}) \\ &> W + C(\Phi^*_{\mathcal{E}}) - C(\Phi_{\mathcal{E}}) > W + C(\Phi^*_{\mathcal{E}}) - W \geq 0, \end{aligned}$$

that is, $C(\Phi^*) > C(\Phi)$, a contradiction. ∎

3.4 The General Case

In the general case of the LaP the distance function, $d(\cdot,\cdot)$, can be any (possibly non-metric) function. For this case, dynamic programming polynomial time optimal algorithms have been proposed for special classes of the object graph G. Bokhari [9] presented an $O(nk^2)$ algorithm for trees and Towesly [73] an $O(nk^3)$ algorithm for series-parallel graphs. Fernádez-Baca, [35] presented an $O(nk^{r+1})$ algorithm for partial r-trees and an $O(nk^{\lceil r/2 \rceil+2})$ algorithm for almost trees with parameter r. Fernández-Baca's algorithms cover the former ones, since trees are either partial 1-trees or almost trees with parameter 0, while series-parallel graphs are partial 2-trees (recall the analogous results for MCP [23]). Furthermore, a lot of branch-and-bound exact algorithms have been proposed for the general case LaP on arbitrary object graphs [8,21,64,70].

4 Local Search Approximation Algorithms

Local search is one of the most successful methods of attacking combinatorial optimization problems. Roughly speaking, local search algorithms start from an arbitrary initial feasible solution and look for an improved one in its neighborhood (i.e. those solutions that do not differ substantially from the initial one). As far as an improved solution exists, it is adopted and the search is repeated from this new solution; otherwise a local optimum is reached (with respect to the chosen neighborhood). In the case of NP-hard problems we are interested on local search obtaining a local optimum solution of cost within some guaranteed factor from the (global) optimum.

Local search approximation algorithms have been also proposed for LaP. Such a local improvement idea was employed, implicitly, in a polynomial time heuristic for the uniform metric LaP, proposed by Milis [65], in the context of distributed software engineering. Boykov et al. [12] proposed, independently, in the context of image processing, the iterative use of the same idea and obtained the first algorithm with constant approximation ratio for the uniform metric LaP. Recently, Gupta and Tardos [47] proposed also a local search algorithm for the truncated linear metric LaP. Note, that both algorithms are based on finding minimum cuts on appropriately constructed graphs.

4.1 The Uniform Metric Case

Consider an arbitrary initial objects' labeling f and let $V_a,\ a \in L$ be the set of objects labeled a in f. The neighborhood, $N_a(f)$, of a labeling f, with respect to a label a, is defined by allowing to any subset of objects to change their labels to a. That is,

$N_a(f) = \{f' | f'(u) = a \text{ if } u \in V'_a \supseteq V_a, \text{ and } f'(u) = f(u), \text{ otherwise}\}.$

A labeling $f' \in N_a(f)$, is called by Boykov et al. [12] an *a-expansion* of f. They also proposed the construction of a appropriate graph G_a such that an optimal a-expansion of f (i.e. the labeling $f' \in N_a(f)$ of minimum cost) can be found by a minimum cut computation in G_a. Instead of their construction, we present here a much simpler one proposed in [65].

The graph $G_a = (V', E')$ is constructed by adding two additional nodes a and $\bar{a}$ to the original object graph, the first representing the label a and the second the labels $L \setminus \{a\}$. Edges are also added between each node $u \in V$ and the nodes a and $\bar{a}$. Thus, $V' = V \cup \{a, \bar{a}\}$ and $E' = E \cup \{(u,a)|u \in V\} \cup \{(u,\bar{a})|u \in V\}$. Then, an optimal a-expansion can be found using next theorem.

Theorem 5. *If the weights of the edges of G_a are*

$$w'(u,\bar{a}) = c(u,a), \quad u \in V,$$

$$w'(u,a) = c(u,f(u)) + \frac{1}{2} \sum_{\substack{(u,v)\in E,\\ f(u)\neq f(v)}} w(u,v), \quad u \in V, \textit{ and}$$

$$w'(u,v) = \begin{cases} w(u,v), & \textit{if } f(u) = f(v), \\ \frac{1}{2}w(u,v), & \textit{otherwise}, \end{cases} \qquad (u,v) \in E,$$

then an optimal a-expansion of a given labeling f is the labeling

$$f'(u) = \begin{cases} a, & \textit{if } u \in V_a \cup V'_a, \\ f(u) & \textit{otherwise}, \end{cases} \qquad u \in V,$$

where V'_a is the set of nodes which remain connected to a by the minimum $a - \bar{a}$ cut of G_a.

Proof. Given a labeling f, any $a - \bar{a}$ cut of G_a is associated with a labeling f' such that $f'(u) = a$, if $u \in V'_a \cup V_a$, and $f'(u) = f(u)$, otherwise. Any labeling f' (and hence the one associated with a minimum $a - \bar{a}$ cut) is clearly an a-expansion of f. It is, therefore, enough to prove that the cost of any $a - \bar{a}$ cut of G_a is equal to the cost of the associated labeling f'.

For the sake of simplicity, consider only two related objects u and v, $(u,v) \in E$. There are two cases for the labeling of u and v in f:

(i) u and v have the same label in f, i.e. $f(u) = f(v)$.
(ii) u and v have different labels in f, i.e. $f(u) \neq f(v)$.

For each case there are four possible cuts/labelings as it is shown in Figure 3, where the weights obtained by the equations of the theorem are also indicated.

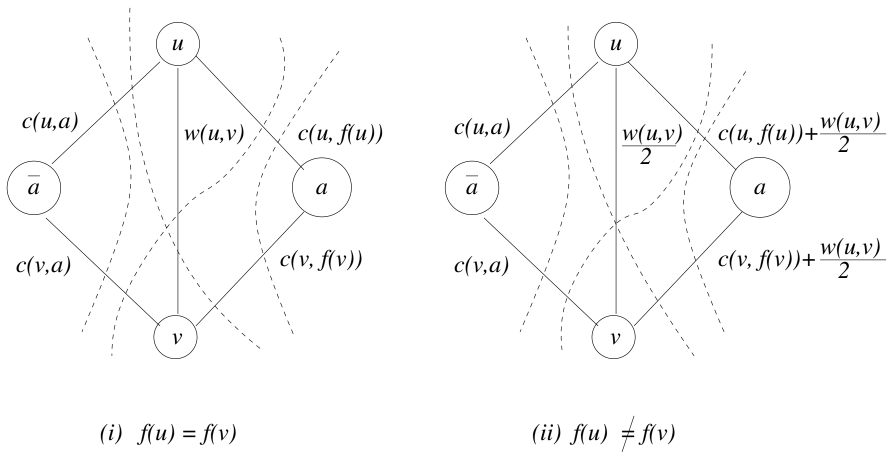

Fig. 3. The construction for the uniform metric case

It is easy to verify that, in both cases, the cost of all possible $a - \bar{a}$ cuts is equal to the cost of the associated labelings. The case when an object is related to two or more other objects is captured by the sum in the second equation. ∎

Given Theorem 5 the following local search algorithm follows.

Algorithm BVZ;

```
choose an initial labeling $f$;
repeat
    for all labels $a \in L$ do
        find an optimal $a$-expansion $f'$ of $f$;
        if $C(f') < C(f)$ then $f = f'$;
until no improvement is possible for any label;
```

Boykov et al. [12] proved the next quite interesting theorem. We present here their proof in a compact form, implied by its generalization in [47].

Theorem 6. **[12]** *The cost of the labeling obtained by algorithm BVZ is within a factor of 2 of the optimum.*

Proof. Let f^* be an optimum labeling. Let also V_a^* be the set of nodes labeled with a in f^*, E_a^* be the set of edges with both of their endpoints in V_a^*, and E_{ab}^* be the set of edges with exactly one of their endpoints in V_a^*.

It is clear that $V = \bigcup_{a \in L} V_a^*$ and $E = E_B \bigcup (\bigcup_{a \in L} E_a^*)$, where $E_B = \bigcup_{a \in L} E_{ab}^*$.

Let f be a labeling and f' be an optimal a-expansion of f. A lower bound on the decrease of the cost $C(f)$ in the movement from f to f' can be obtained by considering the a-expansion f^a of f, which labels a all nodes in V_a^*. That is $C(f) - C(f') \leq C(f) - C(f^a)$. If f is the local optimum labeling obtained by the algorithm BVZ, then this decrease is clearly non-positive, that is $C(f) \leq C(f^a)$.

What changes in the cost $C(f)$, in the movement from f to f^a, is only the assignment cost $A(V_a)$ of the nodes in V_a^* and the separation costs $S(E_a^*)$ and $S(E_{ab}^*)$ of the edges in E_a^* and E_{ab}^*, respectively. In particular, in f^a:

(i) The assignment cost of the nodes in V_a^* is $A^a(V_a^*) = A^*(V_a^*)$ (they are labeled as in f^*).
(ii) The separation cost of the edges in E_a^* is $S^a(E_a^*) = 0$ (their both endpoints are labeled a).
(iii) The separation cost for the edges in E_{ab}^* is $S^a(E_{ab}^*) \leq S^*(E_{ab}^*)$ (since $V_a \supseteq V_a^*$).

Therefore,

$$A(V_a^*) + S(E_a^*) + S(E_{ab}^*) \leq A^*(V_a^*) + S^*(E_{ab}^*).$$

Summing up this inequality, for all $a \in L$, and taking into account that the separation cost of the edges in E_{ab}^* is counted twice in this summation, we obtain

$$A(V) + S(E) - S(E_B) + 2S(E_B) \leq A^*(V) + 2S^*(E_B),$$

and since $S^*(E_B) = S^*(E)$, it follows that

$$A(V) + S(E) + S(E_B) \leq A^*(V) + 2S^*(E), \text{ that is } C(f) \leq 2C(f^*). \quad \blacksquare$$

The complexity of the algorithm BVZ is determined by the number of iterations of its main repeat-until loop. Boykov et al. [12] report that, for applications they have considered in the context of image processing, the algorithm stopped after a few iterations. However, no bound on the number of iterations of the algorithm is known, unless the trivial one of k^n. They also report that most of the labeling improvements were obtained in the first iteration of the algorithm, and this observation agrees with computational results reported in [65], for the evaluation of a heuristic algorithm which performs a single iteration of the algorithm BVZ. More positively, Gupta and Tardos [47] observed that the proof of Theorem 6 can be adapted to show that the algorithm BVZ provides a $2+\epsilon$ approximate solution after $O(\log C^0 + \log \frac{1}{\epsilon})$ iterations of its main loop, where C^0 is the cost of the initial labeling. In fact, the algorithm BVZ can be viewed as a special case of their analogous result for the truncated linear metric case (see next subsection).

Boykov et al. [12,75] have also shown that an optimum a-expansion can be found similarly for any metric distance function. However, in the general metric case, the approximation factor of an analogous local search algorithm is $\rho = 2 \cdot \dfrac{\max\{d(a,b)\}}{\min_{a \neq b}\{d(a,b)\}}$, i.e. the cost of a local optimum solution can be arbitrary far from the optimum one.

4.2 The Truncated Linear Metric Case

In the truncated linear metric case we consider w.l.o.g. that $L = \{1, 2, ..., k\}$, i.e. the labels are the integers 1,2,...,k, and $d(a,b) = \min\{M, |a-b|\}$. Note, that this metric is the most natural and robust non-uniform metric and has direct applications in problems arising in image processing context. For this case, the approximation factor proposed by Boykov et al. for the general metric case becomes $\rho = 2M$. However, Gupta and Tardos [47], generalizing the ideas proposed for the linear and uniform metric cases, presented a local search algorithm obtaining a solution of cost within a factor of 4 of the optimum.

The neighborhood of labeling f is now considered with respect to an interval of the set of labels $\{1, 2, ..., k\}$, instead to a single label. Let $I \subseteq L$ be an interval of the set of labels and V_I be the set of objects with their f labels in I. The neighborhood, $N_I(f)$, of a labeling f, with respect to the interval I, is defined by allowing to any subset of objects to change their labels to a label in I, i.e.

$$N_I(f) = \{f' | f'(u) \in I \text{ if } u \in V'_a \supseteq V_a, \text{ and } f'(u) = f(u), \text{ otherwise}\}.$$

Therefore, a local improvement step results in a labeling f', such that each object either keeps its label or it takes a label in I. Such a labeling f' is called an *I-expansion* of f.

To determine the length of intervals to be considered recall that the (untruncated) linear metric case is polynomial solvable (see Section 3.2). It is, therefore, natural to take intervals of length at most M, since for such an interval the truncated linear metric is reduced to a linear metric. In fact, the length of any interval I is considered to be exactly M, unless I is an initial or final portion of L. Such an interval I can be chosen by simply choosing a random integer r, such that $-M < r < k$, and setting I to be the part of the interval of length M starting from $r + 1$ that lies in L, i.e. $I = \{r + 1, r + 2, ..., r + M\} \cap \{1, 2, ..., k\}$. It is clear that the number of all possible intervals to be considered is $k + M$.

The choice of a random integer r implies a random partition of L into a set of intervals $\mathcal{S} = \{I_0, I_1, ..., I_{\mathcal{S}}\}$, such that each interval I_s is starting either from $[r + s \cdot M + 1]$ $(modk)$ or from 1. However, it is easy to see that choosing r from $\{1, 2, ..., M\}$ is enough to produce all combinatorial distinct partitions of L.

Given a labeling f and an interval of labels $I = \{i + 1, i + 2, ..., j\}$, where $j - i \leq M$, Gupta and Tardos [47] proposed the construction of a graph G_I such that, given two certain terminal nodes s and t of G_I, each $s - t$ cut in G_I is associated with a labeling f' which is an I-expansion of f. We do not present here the details of their construction, which puts together and generalizes the a-expansion idea from the uniform metric case and ideas proposed for the linear metric case (see Section 3.2). However, the construction proposed does not guarantee finding an optimal I-expansion. In fact, the cost of a minimum $s - t$ cut in G_I can be greater than the cost of the associated labeling f'. Fortunately, the gap between these two costs can be bounded as in the next lemma.

Lemma 2. [47] *The gap between the cost of a minimum $s - t$ cut in G_I and the cost of its associated labeling f' is due to a possible overestimation of the separation cost of the labeling f'. In fact, for the edges (u, v) with exactly one of their endpoints having a f' label in I, $d(f'(u), f'(v))$ can be replaced by $d(f'(u), i + 1) + d(i + 1, f'(v))$ and it holds that $d(f'(u), f'(v)) \leq d(f'(u), i + 1) + d(i + 1, f'(v)) \leq 2M$.*

Given this lemma, the following local search algorithm becomes of interest.

```
Algorithm GT;
    choose a random partition $\mathcal{S}$ of $L$;
    choose an initial labeling;
    repeat
        for all $k+m$ possible intervals $I$ do
            find a minimum cut in $G_I$ and the associated labeling $f'$;
            if $C(f') < C(f)$ then $f = f'$ ;
    until no improvement is possible for any interval $I$;
```

Theorem 7. [47] *The cost of the labeling obtained by algorithm GT is within a factor of 4 of the optimum one.*

Proof. Let f^* be the optimum labeling. For an interval I of labels, let V_I^* be the set of nodes with their f^*-labels in I, and E_I^* be the set of edges with both of their endpoints having f^*-labels in I. Let, also, E_{I-}^* (resp. E_{I+}^*) be the set of edges with only the higher (resp. the lower) f^*-label of their endpoints in I.

It is clear that $V = \cup_{I \in \mathcal{S}} V_I^*$, $\bigcup_{I \in \mathcal{S}} E_{I+}^* = \cup_{I \in \mathcal{S}} E_{I-}^* = E_{\mathcal{S}}$ ($E_{\mathcal{S}}$ is used to denote the set of edges whose endpoints have f^*-labels in different intervals of the partition $\mathcal{S}$). Hence, $E = E_{\mathcal{S}} \cup (\cup_{I \in \mathcal{S}} E_I)$.

Let f be a labeling and f' be its I-expansion found by the algorithm. A lower bound on the decrease of the cost $C(f)$ in the movement from f to f' can be obtained by considering the I-expansion f^I of f, which labels all nodes in V_I^* by their f^*-labels. That is $C(f) - C(f') \leq C(f) - C(f^I)$. If f is the local optimum labeling obtained by the algorithm, then this decrease is clearly non-positive, that is $C(f) \leq C(f^I)$.

What changes in the cost $C(f)$, in the movement from f to f^I, is only the assignment cost $A(V_I^*)$ of the nodes in V_I^* and the separation costs $S(E_I^*)$, $S(E_{I+}^*)$ and $S(E_{I-}^*)$ of the edges in E_I^*, E_{I+}^* and E_{I-}^* respectively. For the edges $(u,v) \in S(E_{I+}^*) \cup S(E_{I-}^*)$, assume w.l.o.g. that the f^*-label of their u endpoint is in I. In particular, in f^I:

(i) The assignment cost of the nodes in V_I^* is $A^I(V_I^*) = A^*(V_I^*)$ (they have their f^*-labels).
(ii) The separation cost of the edges in E_I^* is $S^I(E_I^*) = S^*(E_I^*)$ (both of their endpoints have their f^*-labels).
(iii) The separation cost of the edges $(u,v) \in E_{I-}^*$ is, by Lemma 2,

$$S^I(E_{I-}^*) \leq \sum_{(u,v) \in E_{I-}^*} w(u,v)[d(f^*(u), i+1) + d(i+1, f(v))].$$

But, $(u,v) \in E_{I-}^*$ implies $f^*(v) \leq i$. Thus,

$$S^I(E_{I-}^*) \leq \sum_{(u,v) \in E_{I-}^*} w(u,v)[d(f^*(u), f^*(v)) + M] \leq S^*(E_{I-}^*) + M \sum_{(u,v) \in E_{I-}^*} w(u,v).$$

(iv) The separation cost of the edges $(u,v) \in E^*_{I+}$ is, by Lemma 2,

$$S^I(E^*_{I+}) \leq \sum_{(u,v)\in E^*_{I+}} w(u,v)[d(f^*(u),i{+}1){+}d(i{+}1,f(v))] \leq 2M \sum_{(u,v)\in E^+_I} w(u,v).$$

Therefore,

$$A(V^*_I) + S(E^*_I) + S(E^*_{I-}) + S(E^*_{I+})$$

$$\leq A^*(V^*_I) + S^*(E^*_I) + S^*(E^*_{I-}) + M \sum_{(u,v)\in E^*_{I-}} w(u,v) + 2M \sum_{(u,v)\in E^*_{I+}} w(u,v)$$

Summing up last inequality, for all intervals I in the random partition $\mathcal{S}$ of L, and taking into account that the separation cost of the edges in $E^*_{\mathcal{S}}$ is counted twice in this summation, we obtain

$$A(V) + S(E) + S(E_{\mathcal{S}}) \leq A^*(V) + S^*(E) + 3M \sum_{(u,v)\in E_{\mathcal{S}}} w(u,v),$$

that is, $C(f) \leq C(f^*) + 3M \sum_{(u,v)\in E_{\mathcal{S}}} w(u,v)$.

Consider now the expected value of $\sum_{(u,v)\in E_{\mathcal{S}}} w(u,v)$. The probability that an edge (u,v) belongs to $E_{\mathcal{S}}$ is equal to the probability that $f^*(u)$ and $f^*(v)$ lie in different intervals of the partition $\mathcal{S}$, which is exactly $\frac{d(f^*(u),f^*(v))}{M}$ (recall how a random partition $\mathcal{S}$ can be obtained). Hence,

$$\mathbf{E}[M \sum_{(u,v)\in E_{\mathcal{S}}} w(u,v)] = M \sum_{(u,v)\in E} \frac{d(f^*(u),f^*(v))}{M} w(u,v) = S^*(E).$$

Therefore, $C(f) \leq C(f^*) + 3S^*(E) \leq 4C(f^*)$. ∎

The algorithm KT terminates after an exponential number of iterations of its main repeat-until loop. However, Gupta and Tardos [47], proved also that this algorithm provides a $4+\epsilon$ approximate solution after $O(\frac{k}{M}(\log C^0 + \log\frac{1}{\epsilon}))$ iterations of its main loop, where C^0 is the cost of the initial labeling.

It is easy to see that algorithm GT reduces to the algorithm BVZ in the case of $M = 1$, and to algorithm L+ in the case of $M \leq k$. In the first case the truncated linear metric becomes a uniform metric and each interval I of labels consists of a single label. In the second case the truncated linear metric becomes a linear metric and a single interval I of labels is considered. However, in both BVZ and L+ algorithms the cost of a minimum cut in the graphs constructed coincides with the cost of the associated labeling. Thus, in the first case an optimal local movement can be found, while in the second case an optimal solution is directly obtained. It is an interesting open question to construct a graph providing an optimal I-expansion for the truncated linear metric case LaP too.

5 LP-Based Approximation Algorithms

Recently, the LP-relaxation method was used to provide approximation algorithms for LaP. Kleinberg and Tardos [58] and Chekuri et al. [19] presented two different LP-relaxations of the problem and they obtained the best known approximation results for several LaP variants. In this section we present these recent results classified according the problem variants.

5.1 The Uniform Metric Case

Kleinberg and Tardos [58] presented an LP-relaxation for the case of uniform LaP, which is a natural adaptation of the (SLP) relaxation proposed by Călinescu et al. for the MCP [14] and studied, also, in [27] and [52].

A labeling f can be represented by 0-1 variables $x(u,a)$, $u \in V$, $a \in L$, such that (i) $x(u,a) = 1$, if $f(u) = a$, and $x(u,a) = 0$, otherwise, and (ii) $\sum_{a \in L} x(u,a) = 1$, for all $u \in V$. Then, the cost of a labeling can be written as

$$C(f) = \sum_{u \in V, a \in L} c(u,a)x(u,a) + \sum_{(u,v) \in E, a \in L} w(u,v)\frac{1}{2}\sum_{a \in L} |x(u,a) - x(v,a)|$$

Using variables $z(e)$, to express the separation cost due to each edge $e = (u,v) \in E$, and variables $z(e,a)$, to express the absolute value $|x(u,a) - x(v,a)|$ for each label $a \in L$, the uniform LaP can be formulated as the following linear integer program.

$$\begin{array}{lllr}
\text{(UIP)}\ \mathit{Minimize} & \sum_{u \in V, a \in L} c(u,a)x(u,a) + \sum_{e \in E} w(e)z(e) & & \\
\mathit{subject\ to} & \sum_{a \in L} x(u,a) = 1, & \forall v \in V & (1) \\
 & z(e) = \frac{1}{2}\sum_{a \in L} z(e,a), & \forall e \in E & (2) \\
 & z(e,a) \geq x(u,a) - x(v,a), & \forall e \in E, \forall a \in L & (3) \\
 & z(e,a) \geq x(v,a) - x(u,a), & \forall e \in E, \forall a \in L & (4) \\
 & x(u,a) \in \{0,1\}, & \forall u \in V, \forall a \in L & (5)
\end{array}$$

Let (ULP) be the LP-relaxation of (UIP) obtained by replacing the integrality constraints (5) with $x(u,a) \geq 0$, $u \in V, a \in L$. Let $\bar{x}(u,a)$, $u \in V, a \in L$ be a fractional optimal solution to (ULP). This fractional solution is rounded to an integer one by the following procedure, where values $\bar{x}(u,a)$ are used as probabilities that object u is labeled a. In fact, objects are labeled in phases.

Rounding procedure:

```
repeat
    choose a label $a$ uniformly at random from $L$;
    choose a real $\theta$ uniformly at random from [0,1];
    for all unlabeled objects $u \in V$ do
        if $\bar{x}(u,a) \geq \theta$ then label $u$ with $a$;
until all objects are labeled;
```

Theorem 8. [58] *The above algorithm is a 2-approximation algorithm for the uniform LaP.*

Proof. Consider first the assignment cost for an object $u \in V$. Until object u is labeled, the probability it is labeled a in a given phase is exactly $\bar{x}(u,a)/k$. Thus, the probability that an unlabeled object u is labeled in a single phase is precisely $1/k$. Further, over all phases the probability that an object u is labeled a is $\mathbf{Pr}[f(u) = a] = \bar{x}(u,a)$.

Consider next the separation cost of an edge $(u,v) \in E$. The probability that objects u and v are labeled with different labels is:

$$\mathbf{Pr}[f(u) \neq f(v)] \leq \frac{\mathbf{Pr}[\text{exactly one of } u,v \text{ is labeled in one phase}]}{\mathbf{Pr}\ [\text{at least one of } u,v \text{ is labeled in one phase}]}$$

$$\leq \frac{\frac{1}{k}\sum_{a\in L}|\bar{x}(u,a) - \bar{x}(v,a)|}{\frac{1}{k}} = \sum_{a\in L} \bar{z}(e,a) = 2\bar{z}(e).$$

Therefore, the expected cost of the integer solution obtained by the rounding procedure is

$$\mathbf{E}[C_R] = \sum_{u\in V, a\in L} c(u,a)\mathbf{Pr}[f(u) = a] + \sum_{(u,v)\in E} w(e)\mathbf{Pr}[f(u) \neq f(v)]$$

$$\leq \sum_{u\in V, a\in L} c(u,a)\bar{x}(u,a) + 2\sum_{(u,v)\in E} w(e)\bar{z}(e) = \bar{A}(V) + 2\bar{S}(E) \leq 2\bar{C}.$$

As it is mentioned in [58], this rounding procedure can be derandomized using the method of conditional probabilities. ∎

It is easy to see that there is an asymptotic lower bound of 2 on the integrality gap of (ULP) relaxation. Consider as objects' graph the complete graph K_k, with $w(e) = 1$, for all edges, and all assignment costs equal to 0, except costs $c(i,i), 1 \leq i \leq k$, which are equal to infinity. The optimum integral solution is: for some i, label all objects $j \neq i$, by label i and the object i by any other label. The cost of this solution is $k-1$, due to separation cost incurred by the $k-1$ edges incident to node i. An optimal fractional solution is $\bar{x}(i,j) = \frac{1}{k-1}$ for all $i \neq j$ and $\bar{x}(i,j) = 0$ for all $i = j$. The cost of this solution is equal to the separation cost of all $k(k-1)/2$ edges of the objects' graph. The separation cost for each edge is $\frac{1}{k-1}$, that is, a total cost of $k/2$. Hence, a lower bound of $2 - 2/k$ on the integrality gap follows.

Although (ULP) relaxation generalizes (SLP) relaxation, proposed by Calinescu et al. for MCP [14], a different rounding scheme is used, due to the presence of assignment costs $c(u,a)$. This results on an integrality gap of 2 for (ULP) in contrast with the $1.5 - 1/k$ approximation factor and the $8/(7 + \frac{1}{k-1})$ lower bound on the integrality gap of (SLP).

Chekuri et al. [19] proposed another natural integer programming formulation of the general case LaP. Similarly with (UIP), variables 0-1 variables $x(u,a),\ u \in V,\ a \in L$, are also used to express a labeling f. However, instead of variables z_e and z_{ea} in (UIP), they use 0-1 variables $x(u,a,v,b)$ to express directly the labeling of the endpoints u,v of an edge $(u,v) \in E$, i.e. $x(u,a,v,b) = 1$, if $f(u) = a$ and $f(v) = b$, and $x(u,a,v,b) = 0$, otherwise. Thus, the general case LaP can be formulated as following.

$$\text{(GIP)} \quad \mathit{Minimize} \sum_{u \in V, a \in L} c(u,a)x(u,a) + \sum_{\substack{(u,v)\in E\\ a,b \in L}} w(u,v)d(a,b)x(u,a,v,b)$$

$$\mathit{subject\ to} \sum_{a \in L} x(u,a) = 1, \forall u \in V \tag{1}$$

$$\sum_{b \in L} x(u,a,v,b) - x(u,a) = 0,\ \forall u \in V, \forall (u,v) \in E, \forall a \in L \tag{2}$$

$$x(u,a,v,b) - x(v,b,u,a) = 0, \forall u,v \in V, \forall a,b \in L \tag{3}$$

$$x(u,a) \in \{0,1\}, \qquad \forall u \in V, \forall a \in L \tag{4}$$

$$x(u,a,v,b) \in \{0,1\}, \qquad \forall u,v \in V, \forall a,b \in L \tag{5}$$

Constraints (2) force consistency in the edge variables: if $f(u) = a$ and $f(v) = b$, then $x(u,a,v,b)$ is forced to be 1. Constraints (3) express the fact that $x(u,a,v,b)$ and $x(v,b,u,a)$ refer to the same edge. Let (GLP) be the LP-relaxation of (GIP) obtained by relaxing the integrality constraints (4) and (5) and allowing variables $x(u,a)$ and $x(u,a,v,b)$ to take any non-negative value. Observe that constraints (2) are quite important for this relaxation.

For the rest of this subsection, consider (L,d) in (GLP) to be the uniform metric. It is clear that variables $x(u,a)$ determine completely the cost of both (GLP) and (ULP). Let $\bar{x}(u,a)$ be an optimal fractional solution to (GLP) and consider the cost of both (GLP) and (ULP) on $\bar{x}(u,a)$. The assignment cost for both (GLP) and (ULP) is clearly the same. The contribution of an edge $(u,v) \in E$ to the separation cost of (ULP) is

$$\frac{1}{2}\sum_{a\in L} w(u,v)|\bar{x}(u,a) - \bar{x}(v,a)| = \frac{1}{2}\sum_{a \in L} w(u,v)|\sum_{b\in L}\bar{x}(u,a,v,b) - \sum_{b \in L}\bar{x}(v,a,u,b)|$$

$$\leq \frac{1}{2}\sum_{a\in L} w(u,v)|\sum_{b\in L, b\neq a}\bar{x}(u,a,v,b) + \sum_{b\in L, b\neq a}\bar{x}(v,a,u,b)|$$

$$= \frac{1}{2}\sum_{a\in L}\sum_{b \in L, b\neq a} 2w(u,v)\bar{x}(u,a,v,b) = \sum_{a\in L}\sum_{b\in L} w(e)d(a,b)\bar{x}(u,a,v,b).$$

The last term is the contribution of the edge $(u,v) \in E$ to the separation cost of (GLP). Hence, the cost of (ULP) on $\bar{x}(u,a)$ is no more than the cost of (GLP). Therefore, applying the rounding procedure proposed for (ULP), to the fractional solution $\bar{x}(u,a)$ to (GLP) yields the same approximation factor.

Moreover, the K_k graph instance used to show the $2 - 2/k$ lower bound on integrality gap of (ULP), can be also used to show the same lower bound on the integrality gap of (GLP) for the uniform metric LaP.

5.2 The Linear Metric Case

The (GLP) relaxation of Chekuri et al. [19] obtains also another optimal polynomial algorithm for the linear metric case LaP, i.e. the cost of an optimal solution to (GLP) is equal to the cost of an optimal solution to (GIP). In fact, the following procedure is proposed to round an optimal solution $\bar{x}(u,a)$ to (GLP) to an integer optimal one for the linear metric LaP.

Rounding procedure:

```
choose a real $\theta$ uniformly at random from [0,1];
for all $u \in V$ do
    for all $a = 1, 2, ..., k$ do
        $h(u,a) = \sum_{b=1}^{a} \bar{x}(u,b)$;
    if $\theta \leq h(u,1)$ then $f(u) = 1$;
    for all $a = 2, 3, ..., k$ do
        if $h(u,a-1) < \theta \leq h(u,a)$ then $f(u) = a$;
```

It is clear that this rounding procedure (i) labels each object once, since for each object u a unique label a satisfies $h(u,a-1) < \theta \leq h(u,a)$, and (ii) labels all objects, since $h(u,k) = 1$. In fact, this rounding procedure can be easily generalized to the case where the labels are arbitrary points on the real line.

Theorem 9. [19] *The integrality gap of (GLP) relaxation for the linear metric case is 1.*

Proof. It is enough to prove that the expected cost of the labeling f obtained by the rounding procedure is no more than the cost of (GLP) on $\bar{x}(u,a)$.

First, it can be readily verified that for an object $a \in V$, $\mathbf{Pr}[f(u) = a] = \bar{x}(u,a)$.

It is, also, proven in [19] that for an edge $(u,v) \in E$,

$$\mathbf{E}[d(f(u), f(v))] = \sum_{a \in L} |h(u,a) - h(v,a)| \leq \sum_{a,b \in L} d(a,b)\bar{x}(u,a,v,b).$$

Therefore, the expected cost of the labeling obtained by the rounding procedure is

$$\begin{aligned}\mathbf{E}[C_R] &= \sum_{u \in V, a \in L} c(u,a)\mathbf{Pr}[f(u) = a] + \sum_{(u,v) \in E} w(u,v)\mathbf{E}[d(f(u), f(v))] \\ &\leq \sum_{u \in V, a \in L} c(u,a)\bar{x}(u,a) + \sum_{(u,v) \in E, a,b \in L} w(u,v)d(a,b)\bar{x}(u,a,v,b) \\ &= \bar{A}(V) + \bar{S}(E) \leq \bar{C}.\end{aligned}$$

■

Theorem 9 holds also for any strictly convex distance function d on the labels $1, 2, 3, ..., k$, on the integer line [19], that is $d(a, b) = g(|a - b|)$, where g is convex and increasing. Recall that such a distance function is not a metric ((L, d) is a metric iff g is concave and increasing). An example of such a function is the quadratic linear function $d(a, b) = |a - b|^2$, which is of particular interest in some image processing applications [51,61]. The same result on convex functions was also shown by Ishikawa and Geiger [51] in the context of image processing. In fact, they have shown that the graph construction proposed for the linear metric case (see Section 3.2) can be extended for convex distance functions to obtain the optimal solution via minimum cuts.

5.3 The Truncated Linear Metric Case

The (GLP) relaxation of Chekuri et al. [19] obtains also a $2 + \sqrt{2}$- approximation algorithm for the truncated linear metric LaP, thus improving the $4 + \epsilon$-approximation algorithm of Kleinberg and Tardos [58] presented in Section 4.2. For this case they proposed the following rounding procedure which is a natural generalization of the rounding procedures for the uniform and the linear metric cases. The fractional solution $\bar{x}$ is rounded again using only the variables $\bar{x}(u, a)$. The rounding is carried out in phases, as in the uniform metric case. However, in each phase, objects are now labeled by labels drawn from an interval of labels rather than labeled by the same label. Moreover, if two objects u and v are labeled in different phases, then it is assumed that $d(f(u), f(v)) = M$. Once an interval of labels is chosen, the rounding within each phase is similar to the linear case one. An integer parameter $M' \geq M$, is used to determine the starting point of an interval as well as its length. The analysis guides the choice of M' to obtain the best approximation factor.

Rounding procedure:
```
repeat
    choose a real $\theta$ uniformly at random from [0,1];
    choose an integer $l$ uniformly at random from $[-M' + 2, k]$;
    for all unlabeled $u \in V$ do
        for all $a \in I_l = [l, l + M' - 1]$ do
            if $\sum_{b=l}^{a-1} \bar{x}(u, b) < \theta \leq \sum_{b=l}^{a} \bar{x}(u, b)$ then $f(u) = a$;
until all objects are labeled ;
```

Theorem 10. [19] *The above algorithm is a randomized $2 + \sqrt{2}$-approximation algorithm for the truncated linear metric LaP.*

Proof. Consider an iteration of the rounding procedure and an object u not yet labeled. The probability of labeling u by a in this iteration is exactly $\bar{x}(u, a)$, if $a \in I_l$ and zero otherwise. The number of all possible intervals is $k + M' - 1$ and the number of intervals that contain a is M'. Hence, the probability of labeling u by a in this iteration is $\bar{x}(u, a)M'/(k + M' - 1)$. Moreover, the probability of labeling u by some label in an iteration is $M'/(k+M'-1)$. Therefore, the probability that an u is labeled a, over all phases, is $\mathbf{Pr}[f(u)=a]=\bar{x}(u,a)$.

It is proven in [19] that the expected value of the distance $d(f(u),f(v))$ satisfies the inequality

$$\mathbf{E}[d(f(u),f(v))] \le (2+\max\{\frac{2M}{M'},\frac{M'}{M}\}) \sum_{a,b\in L} d(a,b)\bar{x}(u,a,v,b).$$

Hence, the expected cost of the labeling obtained by the rounding procedure is

$$\mathbf{E}[C_R] = \bar{A}(V) + (2+\max\{\frac{2M}{M'},\frac{M'}{M}\})\bar{S}(E) \le (2+\max\{\frac{2M}{M'},\frac{M'}{M}\})\bar{C}.$$

The analysis proposed can be generalized for the case of a real number M', and by choosing $M'=\sqrt{2}M$, the theorem follows. ∎

As it is mentioned in [19] a similar analysis can be also applied in the case of the truncated quadratic linear function $d(a,b)=\min\{M,|a-b|^2\}$ (which is not a metric). In fact, by choosing $M'=\sqrt{M}$, it can be shown that there is a randomized $O(\sqrt{M})$-approximation algorithm for this case.

5.4 The General Metric Case

Kleinberg and Tardos in [58], presented also an $O(\log k \log\log k)$ approximation algorithm for the general metric case LaP. Their algorithm is based on Bartal's result [3,4] that a given metric can be probabilistically approximated by a hierarchically well-separated tree metric (HST metric). Based on this result they proposed an LP-relaxation for the case of HST metrics. This relaxation, called (TLP), is a natural generalization of their (ULP) relaxation for the uniform metric case. They also proposed a randomized procedure for rounding a fractional optimal solution to (TLP) yielding an $O(1)$ integrality gap for HST metrics. Since any general metric can be approximated by an HST metric with an $O(\log k \log\log k)$ distortion [4,18], their result follows.

Chekuri et al. [19] presented a different way to obtain the same result by using directly their (GLP) relaxation. First, they use an optimal solution, $\bar{x}$, to (GLP) to identify a deterministic HST metric approximation, $d_{\bar{T}}$, of the given metric d, such that the cost of $\bar{x}$ on $d_{\bar{T}}$ is bounded. This can be done by using next proposition of Charikar et al. [17].

Proposition 1. [17] *Let d be a metric on L and g be a non-negative function over L. Then d can be approximated deterministically by an HST metric d_T such that*

$$\sum_{a,b} g(a,b)\cdot d_t(a,b) \le O(\log k \log\log k) \sum_{a,b} g(a,b)\cdot d(a,b).$$

Applying Proposition 1, with $g(a,b) = \sum_{(a,b)\in L} w(u,v)\bar{x}(u,a,v,b),\ a,b \in L$, results in a HST metric $d_{\bar{T}}$. The solution $\bar{x}$ to (GLP) on d is, also, a feasible solution to (GLP) on the HST metric $d_{\bar{T}}$, since only the metric over the labels changes. Let $C_{d_{\bar{T}}}$ and $\bar{C}$ be the costs of $\bar{x}$ on d and $d_{\bar{T}}$, respectively. Then, by Proposition 1, it follows that $C_{d_{\bar{T}}} \leq O(\log k \log\log k)\bar{C}$.

Therefore, it is enough to round the optimal fractional solution $\bar{x}$ to (GLP) to an integral solution, provided that the cost of this integral solution on the HST metric $d_{\bar{T}}$ is within an $O(1)$ factor of $C_{d_{\bar{T}}}$. Chekuri et al. [19] claim that their (GLP)-relaxation, for HST metrics, is at least as strong as the (TLP)-relaxation of Kleinberg and Tardos, i.e. for the fractional solution $\bar{x}$, the cost of (GLP) on HST metrics is no more than the cost of (TLP). Therefore, the rounding procedure proposed by Kleinberg and Tardos for (TLP) [58], can be used to get, this way, an $O(\log k \log\log k)$-approximation algorithm.

Theorem 11. **[58,19]** *There is a randomized $O(\log k \log\log k)$-approximation algorithm for the general metric LaP.*

Recently, Fakcharoenphol et al. [34] showed that any general metric can be approximated by an HST metric with an $O(\log k)$ distortion, improving the previous bound of $O(\log k \log\log k)$ distortion on which Theorem 11 is based. Moreover, they showed that their result is existentially tight, i.e. there exist metrics where any HST metric approximation must have $(\log n)$ distortion. Using this result, the rounding procedure proposed by Kleinberg and Tardos for (TLP) [58] leads directly to the next theorem.

Theorem 12. **[34,58]** *There is a randomized $O(\log k)$-approximation algorithm for the general metric LaP.*

Moreover, Archer et al. [1] presented an $O(\log n)$ approximation algorithm for the general metric LaP. Their algorithm uses the same (GLP) relaxation of Chekuri et al. [19] and also approximates the general metric by a HST metric. Instead of using a special relaxation for tree metrics, as Kleinberg and Tardos [58], they propose a rounding method which does not increases the solution's cost for a tree metric. This rounding method provides also an alternate proof to the fact that the (GLP) relaxation of Chekuri et al. [19] has an integral solution for tree metrics (this fact is also mentioned in [19]). However, next theorem outperforms Theorem 12 only if $k \gg n$.

Theorem 13. **[1]** *There is a randomized $O(\log n)$-approximation algorithm for the general metric LaP.*

6 Conclusion

In Table 1 we summarize the known results for the problems studied in this chapter. They are classified into combinatorial results, which are based on minimum cut algorithms, and LP-based results, which are based on LP-relaxations and randomized rounding. For the class of combinatorial algorithms we give in Table

Table 1. Summary of results

Problem	Combinatorial approximation	LP relaxation	Intergality Gap	
			Upper bound	Lower bound
MCP	2-2/k [28]	(SLP) [14]	3/2 - 1/k [14] 1.3438 [52]	$8/(7+\frac{1}{k-1})$ [38]
0-EP		(MLP) [54]	$O(\log k)$ [15]	$\Omega(\sqrt{\log k})$ [15]
Linear LaP	optimal [60, 10, 75, 51]	(GLP) [19]	1 [19]	
Quadratic linear LaP Convex linear LaP	optimal [51]	(GLP)	1 [19]	
Uniform LaP	2^* [12] [47] $2+\epsilon$ [47]	(ULP) [58] (GLP)	2 [58] 2 [19]	
Truncated linear LaP	4^*, $4+\epsilon$ [47]	(GLP)	$2+\sqrt{2}$ [19]	
Truncated quadratic linear LaP		(GLP)	$O(\sqrt{M})$ [19]	
Metric LaP		(TLP) [58] (GLP) (TLP) (GLP)	$O(\log k \log\log k)$ [58] $O(\log k \log\log k)$ [19] $O(\log k)$ [34] $O(\log n)$ [1]	see point (iii) below

* exponential (local search) algorithm

1 the known approximation factors, while for the class of LP-based algorithms we give the LP-relaxations used and the known upper bounds (i.e. approximation factors) and lower bounds on their integrality gap. Recall that all problems in the table are in the Max SNP-hard class and, hence, there is no PTAS for any of them, unless P=NP.

Recent activity on the problems included in Table 1 has led to a significant progress on understanding their approximability though we are still far from resolving the issue. Many open questions can be picked up by looking any entry of Table 1 (even the filled ones). These questions can be classified into three directions.

(i) Closing the gap between the upper and lower bound on the integrality gap of the LP-relaxations proposed is an open question for most of our problems. This can be done either by obtaining problem instances yielding stronger lower bounds (any lower bound if no one is known) or by providing improved rounding procedures yielding better approximation factors or both.

(ii) LP is in P, but the running time of LP algorithms is very high. Therefore, it will be nice to have combinatorial algorithms with approximation factors as close as possible to those of LP-based algorithms. It will be quite interesting to have such an algorithm for any of the empty entries in the corresponding column of Table 1 or even algorithms yielding better approximation factors than the known ones. Note that at least two different 2-approximation algorithms are known for the uniform LaP: an exponential local search one and an LP-based one which, moreover, is obtained through two different LP-relaxations. These results provide a strong evidence that the following conjecture holds.

Conjecture 1. There is a combinatorial (polynomial) 2-approximation algorithm for the uniform LaP.

(iii) Resolving either point (i) or (ii) does not close the approximability question for some problem, unless the approximation factor achieved meets some known approximation threshold. A significant progress to this direction is a recent inapproximability result by Chuzhoy and Naor [24] for the general metric LaP. In fact, they showed that there is no constant factor approximation for the general metric LaP if P≠NP, and also that the problem is $\Omega(\sqrt{\log n})$-hard to approximate if NP $\not\subseteq$ DTIME($n^{poly(\log n)}$), i.e. if NP has no quasi-polynomial time algorithms. It will be of great interest to have such a negative result for any other of our problems. Taking into account again the results for the uniform LaP in Table 1, what we can possibly expect in this direction is a negative result like the following.

Conjecture 2. There is no $(2-\epsilon)$-approximation algorithm for the uniform LaP.

References

1. A. Archer, J. Fakcharoenphol, C. Harrelson, R. Krauthgamer, K. Talwar and E. Tardos, *Approximate classification via earthmover metrics*, In Proc. 17th SODA (2004) 1079-1087.
2. S. Arora, D. R. Karger and Marek Karpinski, *Polynomial time approximation schemes for dense instances of NP-hard problems*, In Proc. 27th STOC (1995) 284-293.
3. Y. Bartal, *Probabilistic approximations of metric spaces and its algorithmic applications*, In Proc. 37th FOCS (1996) 184-193.
4. Y. Bartal, *On approximating arbitrary metrics by tree metrics*, In Proc. 30th STOC (1998) 161-168.
5. D. Bertsimas, C. P. Teo and R. Vohra, *Nonlinear formulations and improved randomized approximation algorithms for multicut problems*, In Proc. 4th IPCO (1995) 29-39.
6. J. Besag, *Spatial interaction and the statistical analysis of lattice systems*, Journal Royal Statistics Society B 36 (1974).
7. J. Besag, *On the statistical analysis of dirty images*, Journal Royal Statistics Society B 48 (1986) 259-302.
8. A. Billionnet, M. C. Costa and A.Sutter, *An efficient algorithm for a task allocation problem*, J.ACM 39 (1992) 502-518.
9. S. H. Bokhari, *A sortest tree algorithm for optimal assignments across space and time in a distributed processor system*, IEEE Trans. Soft. Eng. SE-7 (1981) 583-589.
10. Y. Boykov, O. Veksler and R. Zabih, *Markov random fields with efficient approximations*, In Proc. IEEE International Conference on Computer Vision and Pattern Recognition (1998) 648-655.
11. Y. Boykov, O. Veksler and R. Zabih, *A New Algorithm for Energy Minimization with Discontinuities* In Proc. 2nd International Workoshop Energy Minimization Methods in Computer Vision and Pattern Recognition, EMMCVPR'99, Lecture Notes in Computer Science LNCS-1654 (1999) 205-220.
12. Y. Boykov, O. Veksler and R. Zabih, *Fast Approximate Energy Minimization via Graph Cuts*, In Proc. 7th IEEE International Conference on Computer Vision, ICCV'99 Vol. 1 (1999) 377-384.

13. R. E. Burkard, E. Cela, P.M. Pardalos, and L.S. Pitsoulis, *The quadratic assignment problem*, In Handbook of Combinatorial Optimization, P.M. Pardalos and D.-Z. Du, eds., Kluwer Academic Publishers (1998) 241-338.
14. G. Calinescu, H. Karloff and Y. Rabani, *An improved approximation algorithm for multiway cut*, In Proc. 30th STOC (1998) 48-52. JCSS 60 (2000) 564-574.
15. G. Calinescu, H. Karloff and Y. Rabani, *Approximation algorithms for the 0-extension problem*, In Proc. 12th SODA (2001) 8-16.
16. S. Chakrabarti, B. Dom and P. Indyk, *Enhanced hypertext categorization using hyperlinks*, In Proc. ACM SIGMOD Intern. Conf. on Management of Data (1998) 307-318.
17. M. Charikar, C. Chekuri, A. Goel and S. Guha, *Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median*, In Proc. 30th STOC (1998) 114-123.
18. M. Charikar, C. Chekuri, A. Goel, S. Guha and S. Plotkin, *Approximating a finite metric by a small number of tree metrics*, In Proc, 39th FOCS (1998) 379-388.
19. C. Chekuri, S. Khanna, J. Naor and L. Zosin, *Approximation algorithms for the metric labeling problem via a new linear programming formulation*, In Proc. 12th SODA (2001) 109-118.
20. R. Chellappa, *Markov Random Fields: Theory and Applications*, Academic Press (1993).
21. M. S. Cherh, G.H. Chen and P. Liu, *An LC branch-and-bound algorithm for the module assignment problem*, Information Processing Leters 32 (1989) 61-71.
22. S. Chopra and L. H. Owen, *Extended formulations of the A-cut problem*, preprint (1994).
23. S. Chopra and M. R. Rao, *On the multiway cut polyhedron*, Networks 21 (1991) 51-89.
24. J. Chuzhoy and J. Naor, *The Hardness of Metric Labeling*, Inn Proc. 45th FOCS (2004) 108-114.
25. F. S. Cohen, *Markov Random Fields for image modeling and analysis*, in Modeleling and Applications of Stochastic Processes, U.B. Desai, ed., Kluwer (1986).
26. W. H. Cunningham, *The optimal multiterminal cut problem*, In DIMACS Series in Discrete Mathematics and Theoretical Computer Science 5 (1991) 105-120.
27. W. H. Cunningham and L. Tang *Optimal 3-terminal cuts and linear programming*, In Proc. 7th IPCO (1999) 114-125.
28. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. Seymour and M. Yannakakis, *The complexity of multiterminal cuts*, Preliminary abstract (1983). In Proc. 24th STOC (1992) 241-251. SIAM J. Computing 23 (1994) 864-894.
29. S. Della Pietra, V. Della Pietra and J. D. Lafferty, *Inducing features of random fields*, IEEE Trans. on Pattern Analysis and Machine Intelligence 19 (1997) 380-393.
30. R. Dubes and A. Jain, *Random field models in image analysis*, J. Applied Statistics 16 (1989).
31. T. G. Dietterich, *Machine-learning research*, AI Magazine 18 (1997) 97-136.
32. P. Erdös and L. A. Székely, *Algorithms and min-max theorems for certain multiway cut problems*, In Integer Programming and Combonatorial Optimization, E. Balas, G. Gornuéjols and R. Kannan edts., Graduate Scholl of Industrial Administation, Garnegie-Mellon Univeristy (1992) 334-345.
33. P. Erdös and L. A. Székely, *On weigthed multiway cuts in tress*, Mathematical Programming 65 (1994) 93-105.
34. J. Fakcharoenphol, S. Rao and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, In Proc. 35th STOC (2003) 448-455.

35. D. Fernádez-Baca, *Allocating modules to processors in a distributed system*, IEEE Trans. Soft. Eng. SE-15 (1989) 1427-1436.
36. W. Fernandez de la Vega and M. Lamari, *The task allocation problem with constant communication*, Rapport de Recherche no. 1048, L.R.I., Université de Paris Sud (1996). To appear in Discrete Applied Mathematics.
37. L. R. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ (1962).
38. A. Freund and H. Karloff, *A lower bound of 8/(7+(1/k)-1) on the integrality ratio of the Calinescu-Karloff-Rabani relaxation for multiway cut*, Information Processing Letters 75 (2000) 43-50.
39. A. M. Frieze and R. Kannan, *The regularity lemma and approximation schemes for dense problems*, In Proc. 37th FOCS (1996) 12-20.
40. N. Garg, V. V. Vazirani and M. Yannakakis, *Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover*, In Proc. 20st ICALP, LNCS-700 (1993) 64-75.
41. N. Garg, V. V. Vazirani and M. Yannakakis, *Approximate max-flow min-(multi)cut theorems and their applications*, SIAM J. Computing 25 (1996) 235-251, also in Proc. 25th STOC (1993) 698-707.
42. N. Garg, V. V. Vazirani and M. Yannakakis, *Multiway cuts in directed and node weighted graphs*, In Proc. 21st ICALP, LNCS-820 (1994) 487-498.
43. S. Geman and D. Geman, *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*, IEEE Trans. on Pattern Analysis and Machine Intelligence 6 (1984) 721-741.
44. F. S. Cohen, *Markov random fields for image modelling and analysis*, In Modeling and Applications od Stochastic Processes, U. B. Desai ed., Kluwer Academic Publishers (1986).
45. O. Goldschmidt and D. S. Hochbaum, *A polynomial algorithm for the k-cut problem for fixed k*, Mathematics of Operations Research 35 (1998) 24-37, also In Proc. 29th FOCS (1988) 445-451.
46. D. Greig, B. T. Potrteous and A. Seheult, *Exact minimum a posteriori estimation of binary images*, Journal Royal Statistics Society B 51 (1989) 271-279.
47. A. Gupta and E. Tardos, *A constant factor approximation algorithm for a class of classification problems*, In Proc. 32th STOC (2000) 652-658.
48. X. He, *An improved algorithm for the planar 3-cut problem*, J. of Algorithms 12 (1991) 23-37.
49. D. S. Hochbaum and D. B. Shmoys, *An $O(|V|^2)$ algorithm for the planar 3-cut problem*, SIAM J. Algebraic and Discrete Methods 6 (1985) 707-712.
50. T. C. Hu, *Integer Programming and Network Flows*, Addison- Wesley Publishing Co. Reading, MA (1969).
51. H. Ishikawa and D. Geiger, *Segmentation by grouping junctions*, In Proc. IEEE International Conference on Computer Vision and Pattern Recognition (1998) 125-131.
52. D. R. Karger, P. N. Klein, C. Stein, M. Thorup and N. E. Young, *Rounding algorithms for a geometric embedding of minimum multiway cut*, In Proc. 29th STOC (1999) 668-678.
53. D. R. Karger and C. Stein, *A new approach to the minimum cut problem*, J. ACM 43(1996) 601-640.
54. A. V. Karzanov, *Minimum 0-extension of grpah metrics*, European J. of Combinatorics 19 (1998) 71-101.
55. A. V. Karzanov, *A combinatorial algorithm for the minimum $(2,r)$-metric problem and some generalizations*, Combinatorica 18 (1998) 549-568.

56. R. Kinderman and J. Snell, *Markov random Fields and their Applications*, AMS Press (1980).
57. P. N. Klein, A. Agrawal, R. Ravi and S. Rao, *Approximation through multicommodity flow*, In Proc. 31st FOCS (1990) 726-737.
58. J. Kleinberg and E. Tardos, *Approximation algorithms for classification problems with pairwise relationships: Metric lableling and Markov random fierlds*, In Proc. 40th FOCS (1999) 14-23.
59. Y. Kopidakis, M. Lamari and V. Zissimopoulos, *On the task assignment problem: Two new efficient heuristic algorithms*, J. of Parallel and Distributed Computing 42 (1997) 21-29.
60. C. H. Lee, D. Lee and M. Kim, *Optimal task assignment in linear array networks* IEEE Transanctions on Computers TSC-41 (1992) 877-880.
61. S. Z. Li, *Markov Random Field Modelling in Computer Vision*, Springer-Veralg (1995).
62. V. M. Lo, *Heuristic algorithms for task assignment in distributed systems*, IEEE Trans. Comp. TSC-37 (1988) 1384-1397.
63. V. F. Magirou, *An improved partial solution to the task assignment and multiway cut problems*, Operations Research Letters 12 (1992) 3-10.
64. V. F. Magirou and J. Z. Milis, *An algorithm for the multiprocessor assignment problem*, Operations Research Letters 8 (1989) 351-356.
65. I. Milis, *Task assignment in distributed systems using network flow methods*, In Proc. Combinatorics and Computer Science Conference 1995, LNCS-1120 (1996) 396-405.
66. I. Milis and V. F. Magirou, *A Lagrangian relaxation algorithm for sparse quadratic assignment problems*, Operations Research Letters 17 (1995) 69-76.
67. J. Naor and L. Zosin, *A 2-approximation algorithm for the directed multiway cut problem*, In Proc. 38th FOCS (1997) 548-553.
68. S. Roy and I. J. Cox, *A maximum-flow formulation of the N-camera stereo correspondence problem*, In Proc. 6th IEEE International Conference on Computer Vision, ICCV'98 (1998) 492-499.
69. H. Saran, V. V. Vazirani, *Finding k-cuts within twice the optimal*, In Proc. 32nd FOCS (1991) 743-751.
70. J. B. Sinclair, *Efficient computation of optimal assignment for distributed tasks*, J. Paral. and Distr. Comput. 4 (1987) 342-362.
71. H. S. Stone, *Multiprocessor scheduling with the aid of network flow algorithms*, IEEE Trans. on Software Engineering SE-3 (1977) 85-93.
72. E. Tardos and V. V. Vazirani, *Improved bounds for the max-flow min-multicut ratio for planar and $K_{r,r}$-free graphs*, Information Processing Letters 47 (1993) 77-80.
73. D. F. Towsley, *Allocating programs containing branches and loops within a multiple processor system*, IEEE Trans. Soft. Eng. SE-12 (1986) 1018-1024.
74. V. V. Vazirani, *Approximation algorithms*, Springer, Berlin (2001).
75. O. Veksler, *Efficient graph-based energy minimization methods in computer vision*, Ph. D. Thesis, Department of Computer Science, Cornell University (1999).

List Scheduling in Order of α-Points on a Single Machine

Martin Skutella

Fachbereich Mathematik, Universität Dortmund, D–44221 Dortmund, Germany
`martin.skutella@uni-dortmund.de`
`http://www.mathematik.uni-dortmund.de/~skutella/`

Abstract. Many approximation results for single machine scheduling problems rely on the conversion of preemptive schedules into (preemptive or non-preemptive) solutions. The initial preemptive schedule is usually an optimal solution to a combinatorial relaxation or a linear programming relaxation of the scheduling problem under consideration. It therefore provides a lower bound on the optimal objective function value. However, it also contains structural information which is useful for the construction of provably good feasible schedules. In this context, list scheduling in order of so-called α-points has evolved as an important and successful tool. We give a survey and a uniform presentation of several approximation results for single machine scheduling with total weighted completion time objective from the last years which rely on the concept of α-points.

1 Introduction

We consider the following single machine scheduling problem. There is a set of n jobs $J = \{1, \ldots, n\}$ that must be processed on a single machines. Each job j has a non-negative integral *processing time* p_j, that is, it must be processed during p_j time units. The machine can process at most one job at a time. Each job j has a non-negative integral *release date* r_j before which it cannot be processed. In *preemptive* schedules, a job may repeatedly be interrupted and continued later. In *non-preemptive* schedules, a job must be processed in an uninterrupted fashion. There may be *precedence constraints* between jobs. If $j \prec k$ for $j, k \in J$, it is required that j is completed before k can start. We denote the completion time of job j by C_j and seek to minimize the *total weighted completion time*: A non-negative *weight* w_j is associated with each job j and the goal is to minimize $\sum_{j \in J} w_j C_j$. In order to keep the presentation simple, we will almost always assume that the processing times p_j are positive integers and that $w_j > 0$, for all jobs $j \in J$. However, all results can be carried over to the case with zero processing times and weights.

In scheduling, it is quite convenient to refer to the respective problems using the standard classification scheme of Graham, Lawler, Lenstra, and Rinnooy Kan [20]. The problems that we consider are special variants of the single machine scheduling problem $1|\, r_j, \mathit{prec}\, (, \mathit{pmtn})\, |\sum w_j C_j$. The entry "1" in the first field indicates that the scheduling environment provides only one machine. The second field can be empty or contains some of the job characteristics r_j, prec, and pmtn, indicating whether there are nontrivial release dates or precedence constraints, and whether preemption is allowed. We put an item in brackets to indicate that we consider both variants of the problem, with and without the corresponding feature. The third field refers to the objective function. We are interested in minimizing the total weighted completion time $\sum w_j C_j$ or, for the special case of unit weights, the *total completion time* $\sum C_j$.

Scheduling with the total weighted completion time objective has recently achieved a great deal of attention, partly because of its importance as a fundamental problem in scheduling, and also because of new applications, for instance, in compiler optimization [9] and in parallel computing [6]. In the last years, there has been significant progress in the design of approximation algorithms for various special cases of the general single machine problems $1|\, r_j, \mathit{prec}\,(, \mathit{pmtn})\,|\sum w_j C_j$; see, e.g., [29, 21, 16, 11, 35, 17, 18, 36, 10, 1, 3].

Recall that a *ρ-approximation algorithm* is a polynomial-time algorithm guaranteed to deliver a solution of cost at most ρ times the optimal value; the value ρ is called *performance guarantee* or *performance ratio* of the algorithm. A *randomized* ρ-approximation algorithm is a polynomial-time algorithm that produces a feasible solution whose *expected* objective function value is within a factor of ρ of the optimal value.

For problems without precedence constraints, we also consider the corresponding *on-line setting* where jobs arrive over time and the number of jobs is unknown in advance. Each job $j \in J$ becomes available at its release date, which is not known in advance; at time r_j, we learn both its processing time p_j and its weight w_j. Even in the on-line setting, the value of the computed schedule is compared to the optimal (off-line) schedule. The derived bounds are called *competitive ratios*. While all randomized approximation algorithms discussed in this paper can be derandomized in the off-line setting without loss in the performance guarantee, there is a significant difference in the on-line setting. It is well-known that randomized on-line algorithms often yield better competitive ratios than any deterministic on-line algorithm can achieve (see, e. g., the positive and negative results on the problem $1|\, r_j \,|\sum C_j$ in Table 1).

The conversion of preemptive schedules to nonpreemptive schedules in order to get approximation algorithms was introduced by Phillips, Stein and Wein [29] and was subsequently also used in [7, 11], among others. Phillips et al. [29], and Hall, Shmoys, and Wein [22] introduced the idea of list scheduling in order of α-points to convert preemptive schedules to non-preemptive ones. Even earlier, de Sousa [46] used this idea heuristically to turn 'preemptive' solutions to a time-indexed linear programming relaxation into feasible nonpreemptive schedules. For $\alpha \in (0, 1]$, the α-point of a job with respect to a preemptive schedule is the first point in time when an α-fraction of the job has been completed. Independently from each other, Goemans [16] and Chekuri, Motwani, Natarajan, and Stein [11] have taken up this idea, and showed that choosing α randomly leads to better results. Later, α-points with individual values of α for different jobs have been used by Goemans, Queyranne, Schulz, Skutella, and Wang [17].

Table 1 summarizes approximation and on-line results based on the concept of α-points which were obtained for the single machine scheduling problems under consideration. These results are compared with the best known approximation and on-line results and also with corresponding hardness results. In the absence of precedence

Table 1. Summary of approximation results and non-approximability results. In all cases, $\varepsilon > 0$ can be chosen arbitrarily small. The bounds marked with one and two stars refer to randomized and deterministic on-line algorithms, respectively. The second column contains the results discussed in this paper while the third column gives the best currently known results. The last column contains the best known lower bounds, that is, results on the hardness of approximation.

Approximation and hardness results for single machine scheduling problems

problem	α-points	best known	lower bounds
$1\|\, r_j \,\|\sum C_j$	1.5820^* [11]	$1+\varepsilon$ [1] 2^{**} [29]	1.5820^* [48, 50] 2^{**} [23]
$1\|\, r_j \,\|\sum w_j C_j$	1.6853^* [17] 2.4143^{**} [16]	$1+\varepsilon$ [1] 2^{**} [3]	1.5820^* [48, 50] 2^{**} [23]
$1\|\, r_j,\, pmtn \,\|\sum w_j C_j$	$\frac{4}{3}^*$ [36]	$1+\varepsilon$ [1] 2^{**} [36]	1.038^* [14] 1.073^{**} [14]
$1\|\, prec \,\|\sum w_j C_j$	$2+\varepsilon$ [35]	2 [21]	?
$1\|\, r_j,\, prec \,\|\sum w_j C_j$	$e+\varepsilon$ [35]		?
$1\|\, r_j,\, prec,\, pmtn \,\|\sum w_j C_j$	$2+\varepsilon$ [35]	2 [21]	?

constraints, polynomial-time approximation schemes have recently been obtained by Afrati et al. [1]; however, list scheduling in order of α-points still leads to the best known randomized on-line algorithms for these problems. For problems with precedence constraints, the concept of α-points either yields the best known approximation result (for the problem $1|\, r_j,\, prec \,|\sum w_j C_j$) or only slightly worse results. It is one of the most interesting open problems in the area of machine scheduling to obtain non-approximability results for these problems [38, 51].

We mention further work in the context of α-points: Schulz and Skutella [37] apply the idea of list scheduling in order of α-points to scheduling problems on identical parallel machines, based on a single machine relaxation and random assignments of jobs to machines. Savelsbergh, Uma, and Wein [32] give an experimental study of approximation algorithms for the problem $1|\, r_j \,|\sum w_j C_j$. In particular, they analyze list scheduling in order of α-points for one single α and for individual values of α for different jobs. They also test the approximation algorithms within a branch and bound scheme and apply it to real world scheduling instances arising at BASF AG in Ludwigshafen. Uma and Wein [49] extend this evaluation and study the relationship between several linear programming based lower bounds and combinatorial lower bounds for the problems $1|\, r_j \,|\sum w_j C_j$. The heuristic use of α-points for resource constrained project scheduling problems was also empirically analyzed by Cavalcante, de Souza, Savelsbergh, Wang, and Wolsey [5] and by Möhring, Schulz, Stork, and Uetz [26].

In this paper, we give a survey and a uniform presentation of the approximation results listed in the second column of Table 1. We start with a description and analysis of simple list scheduling heuristics for non-preemptive and preemptive single machine scheduling in Section 2. The concept of α-points is introduced in Section 3. In Section 4 we review the result of Chekuri et al. for the problem $1|\, r_j \,|\sum C_j$. Time-indexed LP relaxations for general constrained single machine scheduling together with results on their quality are discussed in Section 5. In Section 6 we review related results of Schulz [34] and Hall, Schulz, Shmoys, and Wein [21] on list scheduling in order of LP completion times. The approximation results of Goemans et al. [17] and Schulz and Skutella [36] for single machine scheduling with release dates are discussed in Section 7. The results of Schulz and Skutella [35] for the problem with precedence constraints are presented in Section 8. Finally, the application of these results to the on-line setting is discussed in Section 9.

2 Non-preemptive and Preemptive List Scheduling

In this section we introduce and analyze non-preemptive and preemptive list scheduling on a single machine for a given set of jobs with release dates and precedence constraints. We consider both the non-preemptive and the preemptive variant of the problem and argue that the class of list schedules always contains an optimal schedule. Finally we discuss simple approximations based on list scheduling. Many results presented in this section belong to the folklore in the field of single machine scheduling.

Consider a list representing a total order on the set of jobs which extends the partial order given by the precedence constraints. A straightforward way to construct a feasible non-preemptive schedule is to process the jobs in the given order as early as possible with respect to release dates. This routine is called LIST SCHEDULING and a schedule constructed in this way is a (non-preemptive) *list schedule*.

The first result for LIST SCHEDULING in this context was achieved by Smith [45] for the case without release dates or precedence constraints and is known as *Smith's ratio rule*:

Theorem 1. LIST SCHEDULING *in order of non-decreasing ratios p_j/w_j gives an optimal solution for $1|\,|\sum w_j C_j$ in $O(n \log n)$ time.*

Proof. Since all release dates are zero, we can restrict to schedules without idle time. Consider a schedule S with two successive jobs j and k and $p_j/w_j > p_k/w_k$. Exchanging j and k in S leads to a new schedule S' whose value differs from the value of S by $w_j p_k - w_k p_j < 0$. Thus, S is not optimal and the result follows. The running time of LIST SCHEDULING in order of non-decreasing ratios p_j/w_j is dominated by the time needed to sort the jobs, and is therefore $O(n \log n)$. □

For the special case of unit weights ($w_j \equiv 1$), Smith's ratio rule is sometimes also refered to as the *shortest processing time rule* (*SPT-rule*). We always assume that jobs are numbered such that $p_1/w_1 \leqslant \cdots \leqslant p_n/w_n$; moreover, whenever we talk about LIST SCHEDULING in order of non-decreasing ratios p_j/w_j, we refer to this sequence of jobs.

Depending on the given list and the release dates of jobs, one may have to introduce idle time when one job is completed but the next job in the list is not yet released. On the other hand, if preemptions are allowed, it does not make sense to leave the machine idle while another job at a later position in the list is already available (released) and waiting. We would better start this job and preempt it from the machine as soon as the next job in the list is released. In PREEMPTIVE LIST SCHEDULING we process at any point in time the first available job in the list. The resulting schedule is called *preemptive list schedule*. Special applications of this routine have been considered, e. g., by Hall et al. [21] and by Goemans [15, 16]. An important property of preemptive list schedules is that whenever a job is preempted from the machine, it is only continued after all available jobs with higher priority are finished.

Since we can assume without loss of generality that $j \prec k$ implies $r_j \leqslant r_k$, and since the given list is a linear extension of the precedence order, the preemptive list schedule respects precedence constraints and is therefore feasible. The following result on the running time of PREEMPTIVE LIST SCHEDULING has been observed by Goemans [16].

Lemma 1. *Given a list of n jobs, the corresponding list schedule can be created in linear time while* PREEMPTIVE LIST SCHEDULING *can be implemented to run in $O(n \log n)$ time.*

Proof. The first part of the lemma is clear. In order to construct a preemptive list schedule in $O(n \log n)$ time we use a priority queue that always contains the currently available jobs which have not been finished yet; the key of a job is equal to its position in the given list. At each point in time we process the top element of the priority queue or leave the machine idle if the priority queue is empty. There are two types of events that cause an update of the priority queue: Whenever a job is completed on the machine, we remove it from the priority queue; when a job is released, we add it to the priority queue. This results in a total of $O(n)$ priority queue operations each of which can be implemented to run in $O(\log n)$ time. □

As a consequence of the following lemma one can restrict to (preemptive) list schedules.

Lemma 2. *Given a feasible non-preemptive (preemptive) schedule,* (PREEMPTIVE) LIST SCHEDULING *in order of non-decreasing completion times does not increase completion times of jobs.*

Proof. The statement for non-preemptive schedules is easy to see. LIST SCHEDULING in order of non-decreasing completion times coincides with shifting the jobs in the given non-preemptive schedule one after another in order of non-decreasing completion times as far as possible to the left (backwards in time).

Let us turn to the preemptive case. We denote the completion time of a job j in the given schedule by C_j^P and in the preemptive list schedule by C_j. By construction, the new schedule is feasible since no job is processed before its release date. For a fixed job j, let $t \geqslant 0$ be the earliest point in time such that there is no idle time in the preemptive list schedule during $(t, C_j]$ and only jobs k with $C_k^P \leqslant C_j^P$ are processed. We denote the set of these jobs by K. By the definition of t, we know that $r_k \geqslant t$, for all $k \in K$. Hence, $C_j^P \geqslant t + \sum_{k \in K} p_k$. On the other hand, the definition of K implies $C_j = t + \sum_{k \in K} p_k$ and therefore $C_j \leqslant C_j^P$. □

In PREEMPTIVE LIST SCHEDULING a job is only preempted if another job is released at that time. In particular, there are at most $n-1$ preemptions in a preemptive list schedule. Moreover, since all release dates are integral, preemptions only occur at integral points in time. Therefore we can restrict to schedules meeting this property. Nevertheless, we will sometimes consider schedules where preemptions can occur at arbitrary points in time, but we always keep in mind that those scheduled can be converted by PREEMPTIVE LIST SCHEDULING without increasing their value.

2.1 Simple Bounds and Approximations

One of the most important techniques for approximating NP-hard machine scheduling problems with no preemptions allowed is the conversion of preemptive schedules to non-preemptive ones. The first result in this direction has been given by Phillips et al. [29]. It is a 2-approximation algorithm for the problem $1|\, r_j \,|\sum C_j$.

Lemma 3. *Consider* LIST SCHEDULING *according to a list $j_1, j_2, \ldots, j_n$. Then the completion time of job j_i, $1 \leqslant i \leqslant n$, is bounded from above by*

$$\max_{k\leqslant i} r_{j_k} + \sum_{k\leqslant i} p_{j_k} \quad .$$

Proof. For fixed i, modify the given instance by increasing the release date of job j_1 to $\max_{k\leqslant i} r_{j_k}$. The completion time of j_i in the list schedule corresponding to this modified instance is equal to $\max_{k\leqslant i} r_{j_k} + \sum_{k\leqslant i} p_{j_k}$. □

It is well known that the preemptive problem $1|\, r_j, \, pmtn \,|\sum C_j$ can be solved in polynomial time by the *shortest remaining processing time rule* (*SRPT-rule*) [4]: Schedule at any point in time the job with the shortest remaining processing time. The value of this optimal preemptive schedule is a lower bound on the value of an optimal non-preemptive schedule. Together with the following conversion result this yields the 2-approximation algorithm of Phillips et al. [29] for $1|\, r_j \,|\sum C_j$.

Theorem 2. *Given an arbitrary feasible preemptive schedule P,* LIST SCHEDULING *in order of non-decreasing completion times yields a non-preemptive schedule where the completion time of each job is at most twice its completion time in P.*

Proof. The proof follows directly from Lemma 3 since both $max_{k\leqslant i} r_{j_k}$ and $\sum_{k\leqslant i} p_{j_k}$ are lower bounds on C_j^P (we use the notation of Lemma 3). □

An instance showing that the job-by-job bound given in Theorem 2 is tight can be found in [29].

2.2 A Generalization of Smith's Ratio Rule to $1|\, r_j, \, pmtn \,|\sum w_j C_j$

A natural generalization of Smith's ratio rule to $1|\, r_j, \, pmtn \,|\sum w_j C_j$ is PREEMPTIVE LIST SCHEDULING in order of non-decreasing ratios p_j/w_j. Schulz and Skutella [36] give the following lower bound on the performance of this simple heuristic.

Lemma 4. *The performance guarantee of* PREEMPTIVE LIST SCHEDULING *in order of non-decreasing ratios* p_j/w_j *is not better than* 2*, even if* $w_j = 1$ *for all* $j \in J$.

Proof. For an arbitrary $n \in \mathbb{N}$, consider the following instance with n jobs. Let $w_j = 1$, $p_j = n^2 - n + j$, and $r_j = -n + j + \sum_{k=j+1}^{n} p_k$, for $1 \leqslant j \leqslant n$. Preemptive list scheduling in order of non-decreasing ratios of p_j/w_j preempts job j at time r_{j-1}, for $j = 2, \ldots, n$, and finishes it only after all other jobs $j-1, \ldots, 1$ have been completed. The value of this schedule is therefore $n^4 - \frac{1}{2}n^3 + \frac{1}{2}n$. The SRPT-rule, which solves instances of $1|\, r_j, \mathit{pmtn} \,|\sum C_j$ optimally, sequences the jobs in order $n, \ldots, 1$. It has value $\frac{1}{2}n^4 + \frac{1}{3}n^3 + \frac{1}{6}n$. Consequently, the ratio of the objective function values of the SPT-rule and the SRPT-rule goes to 2 when n goes to infinity. □

On the other hand, one can give a matching upper bound of 2 on the performance of this algorithm. As pointed out in [36], the following observation is due to Goemans, Wein, and Williamson.

Lemma 5. PREEMPTIVE LIST SCHEDULING *in order of non-decreasing ratios* p_j/w_j *is a* 2*-approximation for* $1|\, r_j, \mathit{pmtn} \,|\sum w_j C_j$.

Proof. To prove the result we use two different lower bounds on the value of an optimal solution Z^*. Since the completion time of a job is always at least as large as its release date, we get $Z^* \geqslant \sum_j w_j r_j$. The second lower bound is the value of an optimal solution for the relaxed problem where all release dates are zero. This yields $Z^* \geqslant \sum_j \bigl(w_j \sum_{k \leqslant j} p_k\bigr)$ by Smith's ratio rule. Let C_j denote the completion time of job j in the preemptive list schedule. By construction we get $C_j \leqslant r_j + \sum_{k \leqslant j} p_k$ and thus

$$\sum_j w_j C_j \;\leqslant\; \sum_j w_j r_j + \sum_j \bigl(w_j \sum_{k \leqslant j} p_k\bigr) \;\leqslant\; 2\, Z^* \ .$$

This concludes the proof. □

In spite of the negative result in Lemma 4, Schulz and Skutella [36] present an algorithm that converts the preemptive list schedule (in order of non-decreasing ratios p_j/w_j) into another preemptive schedule and achieves performance guarantee $4/3$ for $1|\, r_j, \mathit{pmtn} \,|\sum w_j C_j$; see Section 7. The analysis is based on the observation of Goemans [15] that the preemptive list schedule represents an optimal solution to an appropriate LP relaxation.

3 The Concept of α-Points

In this section we discuss a more sophisticated technique that converts feasible preemptive schedules into non-preemptive ones and generalizes the routine given in Theorem 2. The bounds discussed in this section are at the bottom of the approximation results presented below. Almost all important structural insights are discussed here. We are given a single machine and a set of jobs with release dates and precedence constraints. Besides, we consider a fixed feasible preemptive schedule P and the completion time of job j in this schedule is denoted by C_j^P.

For $0 < \alpha \leqslant 1$ the *α-point* $C_j^P(\alpha)$ of job j with respect to P is the first point in time when an α-fraction of job j has been completed, i. e., when j has been processed on the machine for $\alpha\, p_j$ time units. In particular, $C_j^P(1) = C_j^P$ and for $\alpha = 0$ we define $C_j^P(0)$ to be the starting time of job j.

The average over all points in time at which job j is being processed on the machine is called the *mean busy time* of j. In other words, the mean busy time is the average over all α-points of j, i. e., $\int_0^1 C_j^P(\alpha)\, d\alpha$. This notion has been introduced by Goemans [16]. If j is being continuously processed between $C_j^P - p_j$ and C_j^P, its mean busy time is equal to $C_j^P - \frac{1}{2}p_j$. Otherwise, it is bounded from above by $C_j^P - \frac{1}{2}p_j$.

We will also use the following notation: For a fixed job j and $0 \leqslant \alpha \leqslant 1$ we denote the fraction of job k that is completed by time $C_j^P(\alpha)$ by $\eta_k(\alpha)$; in particular, $\eta_j(\alpha) = \alpha$. The amount of idle time that occurs before time $C_j^P(\alpha)$ on the machine is denoted by $t_{\text{idle}}(\alpha)$. The α-point of j can be expressed in terms of $t_{\text{idle}}(\alpha)$ and $\eta_k(\alpha)$:

Lemma 6. *For a fixed job j and $0 \leqslant \alpha \leqslant 1$ the α-point $C_j^P(\alpha)$ of job j can be written as*

$$C_j^P(\alpha) \;=\; t_{\text{idle}}(\alpha) + \sum_{k\in J} \eta_k(\alpha)\, p_k \;\geqslant\; \sum_{k\in J} \eta_k(\alpha)\, p_k \enspace ,$$

where $\eta_k(\alpha)$ denotes the fraction of job k that is completed by time $C_j^P(\alpha)$.

3.1 List Scheduling in Order of $\boldsymbol{\alpha}$-Points

In Theorem 2 we have analyzed LIST SCHEDULING in order of non-decreasing completion times, i. e., in order of non-decreasing 1-points. Phillips et al. [29], and Hall et al. [22] introduced the idea of LIST SCHEDULING in order of non-decreasing α-points $C_j^P(\alpha)$ for some $0 \leqslant \alpha \leqslant 1$. This is a more general way of capturing the structure of the given preemptive schedule; it can even be refined by choosing α randomly. We call the resulting schedule *α-schedule* and denote the completion time of job j in this schedule by C_j^α.

Goemans et al. [17] have further extended this idea to individual, i. e., job-dependent α_j-points $C_j^P(\alpha_j)$, for $j \in J$ and $0 \leqslant \alpha_j \leqslant 1$. We denote the vector consisting of all α_j's by $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. Then, the α_j-point $C_j^P(\alpha_j)$ is also called *$\boldsymbol{\alpha}$-point* of j and the *$\boldsymbol{\alpha}$-schedule* is constructed by LIST SCHEDULING in order of non-decreasing $\boldsymbol{\alpha}$-points; the completion time of job j in the $\boldsymbol{\alpha}$-schedule is denoted by $C_j^{\boldsymbol{\alpha}}$.

For the feasible preemptive schedule P, the sequence of the jobs in order of non-decreasing $\boldsymbol{\alpha}$-points respects precedence constraints since $j \prec k$ implies $C_j^P(\alpha_j) \leqslant C_j^P \leqslant C_k^P(0) \leqslant C_k^P(\alpha_k)$. Therefore the corresponding $\boldsymbol{\alpha}$-schedule is feasible.

To analyze the completion times of jobs in an $\boldsymbol{\alpha}$-schedule, we also consider schedules that are constructed by a slightly different conversion routine which is called $\boldsymbol{\alpha}$-CONVERSION [17]. A similar procedure is implicitly contained in [11, proof of Lemma 2.2].

$\boldsymbol{\alpha}$-CONVERSION:
Consider the jobs $j \in J$ in order of non-increasing $\boldsymbol{\alpha}$-points $C_j^P(\alpha_j)$ and iteratively change the preemptive schedule P to a non-preemptive schedule by applying the following steps:

i) remove the $\alpha_j\, p_j$ units of job j that are processed before $C_j^P(\alpha_j)$ and leave the machine idle during the corresponding time intervals; we say that this idle time *is caused by* job j;
ii) delay the whole processing that is done later than $C_j^P(\alpha_j)$ by p_j;
iii) remove the remaining $(1-\alpha_j)$-fraction of job j from the machine and shrink the corresponding time intervals; *shrinking* a time interval means to discard the interval and move earlier, by the corresponding amount, any processing that occurs later;
iv) process job j in the released time interval $(C_j^P(\alpha_j), C_j^P(\alpha_j)+p_j]$.

Figure 1 contains an example illustrating the action of $\boldsymbol{\alpha}$-CONVERSION. Observe that in the resulting schedule jobs are processed in non-decreasing order of $\boldsymbol{\alpha}$-points and no job j is started before time $C_j^P(\alpha_j) \geqslant r_j$. The latter property will be useful in the analysis of on-line $\boldsymbol{\alpha}$-schedules in Section 9.

Lemma 7. *The completion time of job j in the schedule computed by $\boldsymbol{\alpha}$-*CONVERSION *is equal to*

$$C_j^P(\alpha_j) + \sum_{\substack{k \\ \eta_k(\alpha_j)\geqslant\alpha_k}} \bigl(1+\alpha_k-\eta_k(\alpha_j)\bigr)\, p_k \enspace .$$

Proof. Consider the schedule constructed by $\boldsymbol{\alpha}$-CONVERSION. The completion time of job j is equal to the idle time before its start plus the sum of processing times of jobs that start no later than j. Since the jobs are processed in non-decreasing order of their $\boldsymbol{\alpha}$-points, the amount of processing before the completion of job j is

$$\sum_{\substack{k \\ \alpha_k\leqslant\eta_k(\alpha_j)}} p_k \enspace . \tag{1}$$

The idle time before the start of job j can be written as the sum of the idle time $t_{\text{idle}}(\alpha_j)$ that already existed in the preemptive schedule P before $C_j^P(\alpha_j)$ plus the idle time before the start of job j that is caused in steps i) of $\boldsymbol{\alpha}$-CONVERSION; notice that steps iii) do not create any additional idle time since we shrink the affected time intervals. Each job k that is started no later than j, i. e., such that $\eta_k(\alpha_j) \geqslant \alpha_k$, contributes $\alpha_k\, p_k$ units of idle time, all other jobs k only contribute $\eta_k(\alpha_j)\, p_k$ units of idle time. As a result, the total idle time before the start of job j can be written as

$$t_{\text{idle}}(\alpha_j) + \sum_{\substack{k \\ \alpha_k\leqslant\eta_k(\alpha_j)}} \alpha_k\, p_k + \sum_{\substack{k \\ \alpha_k>\eta_k(\alpha_j)}} \eta_k(\alpha_j)\, p_k \enspace . \tag{2}$$

The completion time of job j in the schedule constructed by $\boldsymbol{\alpha}$-CONVERSION is equal to the sum of the expressions in (1) and (2); the result then follows from Lemma 6. □

Instance:

job j		r_j	p_j	α_j
1	▨	0	2	$\frac{1}{8}$
2	▤	1	2	$\frac{5}{8}$
3	⬡	2	1	$\frac{1}{4}$
4	▦	4	2	$\frac{1}{8}$

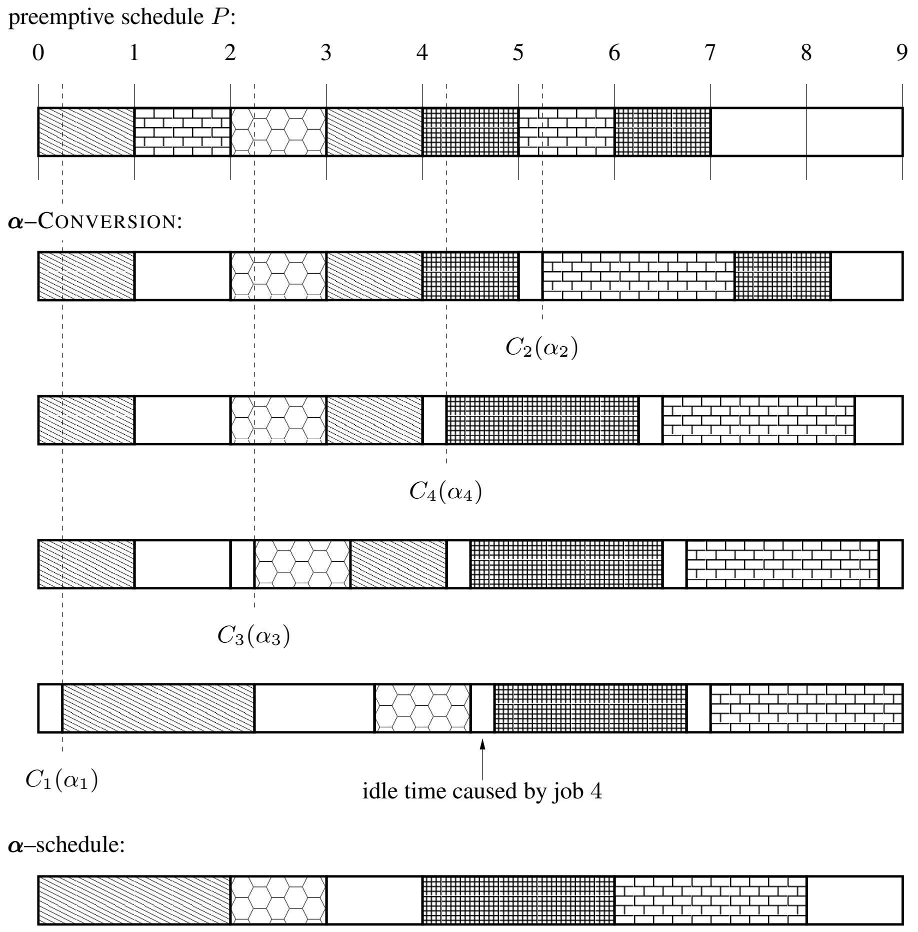

Fig. 1. The conversion of an arbitrary preemptive schedule P to a non-preemptive one by $\boldsymbol{\alpha}$-CONVERSION and by LIST SCHEDULING in order of non-decreasing $\boldsymbol{\alpha}$-points

It follows from Lemma 7 that the completion time C_j of each job j in the non-preemptive schedule constructed by $\boldsymbol{\alpha}$-CONVERSION satisfies $C_j \geqslant C_j^P(\alpha_j) + p_j \geqslant r_j + p_j$, hence is a feasible schedule. We obtain the following corollary.

Corollary 1. *The completion time of job j in an $\boldsymbol{\alpha}$-schedule can be bounded by*

$$C_j^{\boldsymbol{\alpha}} \leqslant C_j^P(\alpha_j) + \sum_{\substack{k \\ \eta_k(\alpha_j) \geqslant \alpha_k}} \bigl(1 + \alpha_k - \eta_k(\alpha_j)\bigr)\, p_k \ .$$

Proof. Since the schedule constructed by $\boldsymbol{\alpha}$-CONVERSION processes the jobs in order of non-decreasing $\boldsymbol{\alpha}$-points, the result follows from Lemma 7 and Lemma 2. □

3.2 Preemptive List Scheduling in Order of $\boldsymbol{\alpha}$-Points

Up to now we have considered non-preemptive LIST SCHEDULING in order of $\boldsymbol{\alpha}$-points. For a vector $\boldsymbol{\alpha}$ we call the schedule that is constructed by PREEMPTIVE LIST SCHEDULING in order of non-decreasing $\boldsymbol{\alpha}$-points *preemptive $\boldsymbol{\alpha}$-schedule*. The completion time of job j in the preemptive $\boldsymbol{\alpha}$-schedule is denoted by $C_j^{\boldsymbol{\alpha}-pmtn}$.

The reader might wonder why we want to convert a preemptive schedule into another preemptive schedule. Later we will interpret solutions to time-indexed LP relaxations as preemptive schedules. However, the value of these schedules can be arbitrarily bad compared to their LP value. The results derived in this section will help to turn them into provably good preemptive schedules.

Again, to analyze preemptive $\boldsymbol{\alpha}$-schedules we consider an alternative conversion routine which we call PREEMPTIVE $\boldsymbol{\alpha}$-CONVERSION and which is a modification of $\boldsymbol{\alpha}$-CONVERSION. The difference is that there is no need for causing idle time by removing the α_j-fraction of job j from the machine in step i). We rather process $\alpha_j\, p_j$ units of job j. Thus, we have to postpone the whole processing that is done later than $C_j^P(\alpha_j)$ only by $(1 - \alpha_j)\, p_j$, because this is the remaining processing time of job j, which is then scheduled in $\bigl(C_j^P(\alpha_j), C_j^P(\alpha_j) + (1 - \alpha_j)\, p_j\bigr]$.

This conversion technique has been introduced by Goemans, Wein, and Williamson [18] for a single α. Figure 2 contains an example illustrating the action of PREEMPTIVE $\boldsymbol{\alpha}$-CONVERSION. Observe that, as in the non-preemptive case, the order of completion times in the resulting schedule coincides with the order of $\boldsymbol{\alpha}$-points in P. Moreover, since the initial schedule P is feasible, the same holds for the resulting schedule by construction.

Lemma 8. *The completion time of job j in the preemptive $\boldsymbol{\alpha}$-schedule can be bounded by*

$$C_j^{\boldsymbol{\alpha}-pmtn} \leqslant C_j^P(\alpha_j) + \sum_{\substack{k \\ \eta_k(\alpha_j) \geqslant \alpha_k}} \bigl(1 - \eta_k(\alpha_j)\bigr)\, p_k \ . \qquad (3)$$

In the absence of nontrivial release dates, the same bound holds for the completion time $C_j^{\boldsymbol{\alpha}}$ of job j in the non-preemptive $\boldsymbol{\alpha}$-schedule.

Proof. Using the same ideas as in the proof of Lemma 7 it can be seen that the right hand side of (3) is equal to the completion time of job j in the schedule constructed by PREEMPTIVE $\boldsymbol{\alpha}$-CONVERSION. The only difference to the non-preemptive setting is that no additional idle time is caused by the jobs. Since the order of completion times in the schedule constructed by PREEMPTIVE $\boldsymbol{\alpha}$-CONVERSION coincides with the order of $\boldsymbol{\alpha}$-points, the bound in (3) follows from Lemma 2.

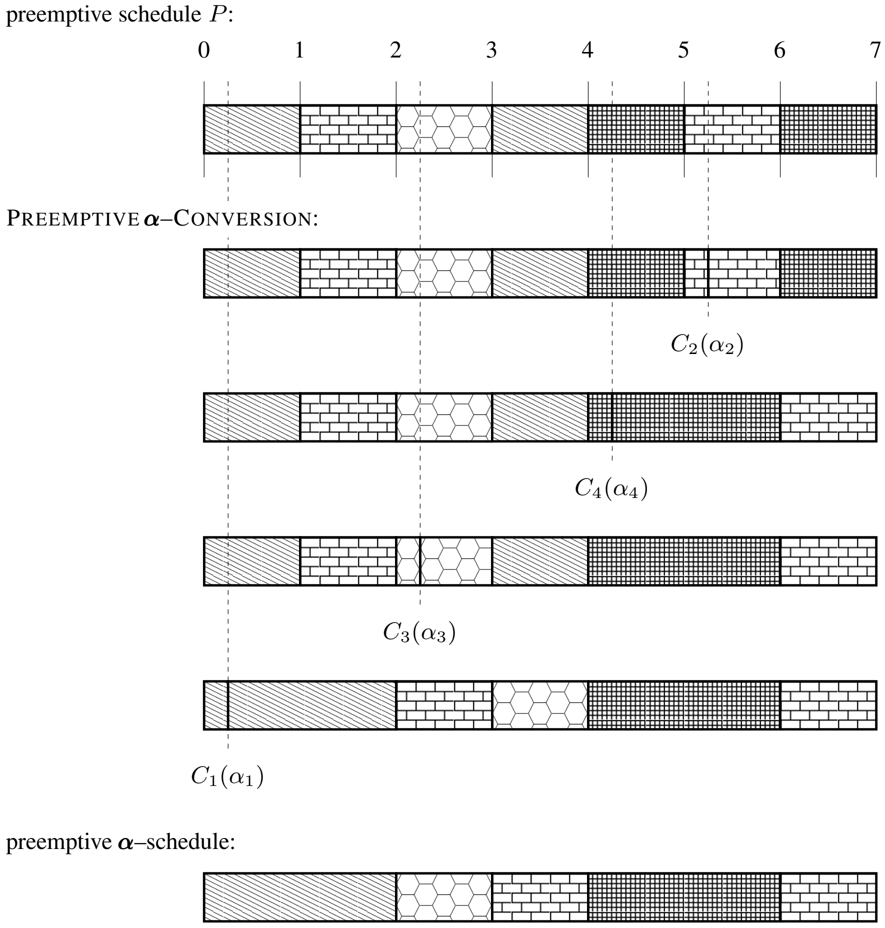

Fig. 2. The conversion of the preemptive schedule P by PREEMPTIVE α-CONVERSION and by PREEMPTIVE LIST SCHEDULING in order of non-decreasing α-points for the instance given in Figure 1

In the absence of nontrivial release dates, PREEMPTIVE LIST SCHEDULING always constructs a non-preemptive schedule that coincides with the non-preemptive list schedule. In particular, $C_j^{\alpha} = C_j^{\alpha-pmtn}$ and the bound given in (3) holds for C_j^{α} too. □

Lemma 6, Corollary 1, and Lemma 8 contain all structural insights in (preemptive) α-schedules that are needed to derive the approximation results presented below.

3.3 Scheduling in Order of α-Points for Only One α

Corollary 1 and Lemma 6 yield the following generalization of Theorem 2 that was first presented in [11]:

Theorem 3. *For each fixed* $0 < \alpha \leqslant 1$, *the completion time of job* j *in the* α*-schedule can be bounded by*

$$C_j^\alpha \leqslant \Big(1 + \frac{1}{\alpha}\Big) C_j^P(\alpha) \leqslant \Big(1 + \frac{1}{\alpha}\Big) C_j^P .$$

Proof. Corollary 1 and Lemma 6 yield

$$\begin{aligned} C_j^\alpha &\leqslant C_j^P(\alpha) + \sum_{\substack{k \\ \eta_k(\alpha) \geqslant \alpha}} \big(1 + \alpha - \eta_k(\alpha)\big)\, p_k \\ &\leqslant C_j^P(\alpha) + \sum_{\substack{k \\ \eta_k(\alpha) \geqslant \alpha}} p_k \\ &\leqslant C_j^P(\alpha) + \sum_{\substack{k \\ \eta_k(\alpha) \geqslant \alpha}} \frac{\eta_k(\alpha)}{\alpha}\, p_k \\ &\leqslant \Big(1 + \frac{1}{\alpha}\Big) C_j^P(\alpha) . \end{aligned}$$

The second bound in Theorem 3 follows from the definition of α-points. □

It is shown in [11] that the bound given in Theorem 3 is tight. The following lemma is an analogue to Theorem 3 for preemptive α-schedules.

Lemma 9. *For each fixed* $0 < \alpha \leqslant 1$, *the completion time of job* j *in the preemptive* α*-schedule can be bounded by*

$$C_j^{\alpha-pmtn} \leqslant \frac{1}{\alpha} C_j^P(\alpha) \leqslant \frac{1}{\alpha} C_j^P .$$

In the absence of nontrivial release dates the same bound holds for the completion time C_j^α *of job* j *in the non-preemptive* α*-schedule.*

Proof. Lemma 8 and Lemma 6 yield

$$\begin{aligned} C_j^{\alpha-pmtn} &\leqslant C_j^P(\alpha) + \sum_{\substack{k \\ \eta_k(\alpha) \geqslant \alpha}} \big(1 - \eta_k(\alpha)\big)\, p_k \\ &\leqslant C_j^P(\alpha) + (1-\alpha) \sum_{\substack{k \\ \eta_k(\alpha) \geqslant \alpha}} p_k \\ &\leqslant C_j^P(\alpha) + \frac{1-\alpha}{\alpha} \sum_{\substack{k \\ \eta_k(\alpha) \geqslant \alpha}} \eta_k(\alpha)\, p_k \\ &\leqslant \frac{1}{\alpha} C_j^P(\alpha) . \end{aligned}$$

In the absence of nontrivial release dates the same bound holds for C_j^α since the result of Lemma 8 can be applied in this case too. □

4 An $e/(e-1)$-Approximation Algorithm for $1|\,r_j\,|\sum C_j$

In this section we present the result of Chekuri et al. [11] on the conversion of preemptive to non-preemptive single machine schedules. Moreover, we discuss a class of instances found by Goemans et al. [17] which shows that this result is tight, that is, it yields a tight worst case bound on the value of an optimal non-preemptive schedule in terms of the optimal value with preemptions allowed.

In Theorem 3 the best choice of α seems to be $\alpha = 1$ yielding the factor 2 bound that has already been determined in Theorem 2. The key insight of Chekuri et al. is that one can get a better (expected) bound by drawing α randomly from $[0,1]$ instead of using only one fixed value of α. The underlying intuition is that the completion time C_j^{α} of a given job j in the α-schedule cannot attain the worst case bound $\left(1+\frac{1}{\alpha}\right) C_j^P(\alpha)$ for all possible choices of α. Moreover, although the worst case bound is bad for small values of α, the overall α-schedule might be good since the total amount of idle time caused by jobs in the first step of $\boldsymbol{\alpha}$-CONVERSION is small and the schedule is therefore short.

Theorem 4. *For each job j, let α_j be chosen from a probability distribution over $[0,1]$ with density function $f(x) = e^x/(e-1)$. Then, the expected completion time $\mathrm{E}[C_j^{\boldsymbol{\alpha}}]$ of job j in the $\boldsymbol{\alpha}$-schedule can be bounded from above by $\frac{e}{e-1}\,C_j^P$, where C_j^P denotes the completion time of job j in the given preemptive schedule P.*

Proof. To simplify notation we denote $\eta_k(1)$ by η_k. For any fixed $\boldsymbol{\alpha}$, Corollary 1 and Lemma 6 yield

$$\begin{aligned}
C_j^{\boldsymbol{\alpha}} &\leqslant C_j^P(\alpha_j) + \sum_{\substack{k\\ \eta_k \geqslant \alpha_k}} \bigl(1+\alpha_k-\eta_k(\alpha_j)\bigr)\,p_k \\
&= C_j^P(\alpha_j) + \sum_{\substack{k\\ \eta_k \geqslant \alpha_k}} \bigl(\eta_k-\eta_k(\alpha_j)\bigr)\,p_k + \sum_{\substack{k\\ \eta_k \geqslant \alpha_k}} (1+\alpha_k-\eta_k)\,p_k \\
&\leqslant C_j^P + \sum_{\substack{k\\ \eta_k \geqslant \alpha_k}} (1+\alpha_k-\eta_k)\,p_k \quad .
\end{aligned}$$

In order to compute a bound on the expected completion time of job j, we integrate the derived bound on $C_j^{\boldsymbol{\alpha}}$ over all possible choices of the random variables α_k, $k \in J$, weighted by the given density function f. Since

$$\int_0^{\eta} f(x)\,(1+x-\eta)\,dx \;=\; \frac{\eta}{e-1}$$

for each $\eta \in [0,1]$, the expected completion time of job j can be bounded by

$$\begin{aligned}
\mathrm{E}[C_j^{\boldsymbol{\alpha}}] &\leqslant C_j^P + \sum_k p_k \int_0^{\eta_k} f(\alpha_k)\,(1+\alpha_k-\eta_k)\,d\alpha_k \\
&= C_j^P + \frac{1}{e-1}\sum_k \eta_k\,p_k \;\leqslant\; \frac{e}{e-1}\,C_j^P \quad .
\end{aligned}$$

This concludes the proof. □

Theorem 4 is slightly more general than the original result in [11]. The only condition on the random choice of the vector $\boldsymbol{\alpha}$ in Theorem 4 is that each of its entries α_j has to be drawn with density function f from the interval $[0,1]$. Chekuri et al. only consider the case where $\alpha_j = \alpha$ for each $j \in J$ and α is drawn from $[0,1]$ with density function f. However, the analysis also works for arbitrary interdependencies between the different random variables α_j, e. g., for jointly or pairwise independent α_j. The advantage of using only one random variable α for all jobs is that the resulting randomized approximation algorithm can easily be derandomized. We discuss this issue in more detail below.

Corollary 2. *Suppose that $\boldsymbol{\alpha}$ is chosen as described in Theorem 4. Then, the expected value of the $\boldsymbol{\alpha}$-schedule is bounded from above by $\frac{e}{e-1}$ times the value of the preemptive schedule P.*

Proof. Using Theorem 4 and linearity of expectations yields

$$\mathrm{E}\Big[\sum_j w_j C_j^{\boldsymbol{\alpha}}\Big] = \sum_j w_j \mathrm{E}[C_j^{\boldsymbol{\alpha}}] \leqslant \frac{e}{e-1} \sum_j w_j C_j^P \ .$$

This concludes the proof. □

Since the preemptive problem $1|\,r_j,\,pmtn\,|\sum C_j$ can be solved in polynomial time by the shortest remaining processing time rule, computing this optimal solution and converting it as described in Theorem 4 is a randomized approximation algorithm with expected performance guarantee $\frac{e}{e-1}$ for $1|\,r_j\,|\sum C_j$.

The variant of this randomized algorithm with only one random variable α for all jobs instead of individual α_j's is of special interest. In fact, starting from a preemptive list schedule P (e. g., the one constructed by the SRPT-rule), one can efficiently compute an α-schedule of least objective function value over *all* α between 0 and 1; we refer to this schedule as the *best α-schedule*. The following proposition, which can be found in [17], yields a deterministic $\frac{e}{e-1}$-approximation algorithm with running time $O(n^2)$ for the problem $1|\,r_j\,|\sum C_j$.

Proposition 1. *For a fixed preemptive list schedule P, there are at most n different (preemptive) α-schedules; they can be computed in $O(n^2)$ time.*

Proof. As α goes from 0 to 1, the α-schedule changes only whenever an α-point, say for job j, reaches a time at which job j is preempted. Thus, the total number of changes in the (preemptive) α-schedule is bounded from above by the total number of preemptions. Since a preemption can occur in the preemptive list schedule P only whenever a job is released, the total number of preemptions is at most $n-1$, and the number of (preemptive) α-schedules is at most n. Since each of these (preemptive) α-schedules can be computed in $O(n)$ time, the result on the running time follows. □

For the weighted scheduling problem $1|\,r_j\,|\sum w_j C_j$, even the preemptive variant is strongly NP-hard, see [24]. However, given a ρ-approximation algorithm for one of the preemptive scheduling problems $1|\,r_j,\,(prec,)\,pmtn\,|\sum w_j C_j$, the conversion technique of Chekuri et al. can be used to design an approximation with performance ratio $\rho \frac{e}{e-1}$ for the corresponding non-preemptive problem. Unfortunately, this does not directly lead to improved approximation results for these problems. In Section 8, however, we present a more sophisticated combination of a 2-approximation algorithm for $1|\, r_j,\, prec,\, pmtn \,|\sum w_j C_j$ with this conversion technique by Schulz and Skutella [35], which yields performance guarantee e for the resulting algorithm.

Another consequence of Corollary 2 is the following result on the power of preemption which can be found in [17].

Theorem 5. *For single machine scheduling with release dates and precedence constraints so as to minimize the weighted sum of completion times, the value of an optimal non-preemptive schedule is at most $\frac{e}{e-1}$ times the value of an optimal preemptive schedule and this bound is tight, even in the absence of precedence constraints.*

Proof. Suppose that P is an optimal preemptive schedule. Then, the expected value of an $\boldsymbol{\alpha}$-schedule, with $\boldsymbol{\alpha}$ chosen as described in Theorem 4, is bounded by $\frac{e}{e-1}$ times the value of the optimal preemptive schedule P. In particular, there must exist at least one fixed choice of the random variable $\boldsymbol{\alpha}$ such that the value of the corresponding non-preemptive $\boldsymbol{\alpha}$-schedule can be bounded from above by $\frac{e}{e-1} \sum_j w_j C_j^P$.

To prove the tightness of this bound we consider the following instance with $n \geqslant 2$ jobs. There is one large job, denoted job n, and $n-1$ small jobs denoted $j = 1, \ldots, n-1$. The large job has processing time $p_n = n$, weight $w_n = 1/n$ and release date $r_n = 0$. Each of the $n-1$ small jobs j has zero processing time, weight $w_j = \frac{1}{n(n-1)}(1 + \frac{1}{n-1})^{n-j}$, and release date $r_j = j$.

The optimal preemptive schedule has job n start at time 0, preempted by each of the small jobs; hence its completion times are: $C_j = r_j$ for $j = 1, \ldots, n-1$ and $C_n = n$. Its objective function value is $(1 + \frac{1}{n-1})^n - (1 + \frac{1}{n-1})$ and approaches $e-1$ for large n.

Now consider an optimal non-preemptive schedule C^* and let $k = \lfloor C_n^* \rfloor - n \geq 0$, so k is the number of small jobs that can be processed before job n. It is then optimal to process all these small jobs $1, \ldots, k$ at their release dates, and to start processing job n at date $r_k = k$ just after job k. It is also optimal to process all remaining jobs $k+1, \ldots, n-1$ at date $k+n$ just after job n. Let C^k denote the resulting schedule, that is, $C_j^k = j$ for all $j \leq k$, and $C_j^k = k+n$ otherwise. Its objective function value is $(1 + \frac{1}{n-1})^n - \frac{1}{n-1} - \frac{k}{n(n-1)}$. Therefore the optimal schedule is C^{n-1} with objective function value $(1 + \frac{1}{n-1})^n - \frac{1}{n-1} - \frac{1}{n}$. Thus, as n grows large, the optimal non-preemptive cost approaches e. □

Unfortunately, the result of Theorem 5 cannot be easily extended to the case of unit weights. The instance discussed in the proof of the theorem relies on the large number of jobs with processing time zero whose weights are small compared to the weight of the large job.

5 Time-Indexed LP Relaxations

As already mentioned in the last section, even the preemptive variants of the scheduling problems $1|\, r_j \,|\sum w_j C_j$ and $1|\, prec \,|\sum w_j C_j$ are NP-hard. In order to give approximation results for these problems, we consider a time-indexed LP relaxation of

$1|\,r_j,\, prec,\, pmtn\,|\sum w_j C_j$ whose optimal value serves as a surrogate for the true optimum in our estimations. Moreover, we interpret a solution to this relaxation as a so-called *fractional preemptive schedule* which can be converted into a non-preemptive one using the techniques discussed in the previous sections.

For technical reasons only, we assume in the following that all processing times of jobs are positive. This is only done to keep the presentation of the techniques and results as simple as possible. Using additional constraints in the LP relaxation, all results presented can be carried over to the case with zero processing times.

The following LP relaxation is an immediate extension of a time-indexed LP proposed by Dyer and Wolsey [13] for the problem without precedence constraints. Here, time is discretized into the periods $(t, t+1]$, $t = 0, 1, \ldots, T$ where $T+1$ is the planning horizon, say $T + 1 := \max_{j\in J} r_j + \sum_{j\in J} p_j$. We have a variable y_{jt} for every job j and every time period $(t, t+1]$ which is set to 1 if j is being processed in that time period and to 0 otherwise. The LP relaxation is as follows:

$$
\begin{array}{llll}
 & \text{minimize} & \displaystyle\sum_{j\in J} w_j C_j^{LP} & \\
 & \text{subject to} & \displaystyle\sum_{t=r_j}^{T} y_{jt} = p_j & \text{for all } j \in J, \quad (4)\\
 & & \displaystyle\sum_{j\in J} y_{jt} \leqslant 1 & \text{for } t = 0, \ldots, T, \quad (5)\\
(LP) & & \displaystyle\frac{1}{p_j}\sum_{\ell=r_j}^{t} y_{j\ell} \geqslant \frac{1}{p_k}\sum_{\ell=r_k}^{t} y_{k\ell} & \text{for all } j \prec k \text{ and } t = 0, \ldots, T, \quad (6)\\
 & & \displaystyle C_j^{LP} = \frac{p_j}{2} + \frac{1}{p_j}\sum_{t=r_j}^{T} y_{jt}\big(t + \tfrac{1}{2}\big) & \text{for all } j \in J, \quad (7)\\
 & & y_{jt} = 0 & \text{for all } j \in J \text{ and } t = 0, \ldots, r_j - 1,\\
 & & y_{jt} \geqslant 0 & \text{for all } j \in J \text{ and } t = r_j, \ldots, T.
\end{array}
$$

Equations (4) say that all fractions of a job, which are processed in accordance with its release date, must sum up to the whole job. Since the machine can process only one job at a time, the machine capacity constraints (5) must be satisfied. Constraints (6) say that at any point $t + 1$ in time the completed fraction of job k must not be larger than the completed fraction of its predecessor j.

Consider an arbitrary feasible schedule P, where job j is being continuously processed between $C_j^P - p_j$ and C_j^P on the machine. Then, the expression for C_j^{LP} in (7) corresponds to the real completion time C_j^P of j, if we assign the values to the LP variables y_{jt} as defined above, that is, $y_{jt} = 1$ if j is being processed in the time interval $(t, t+1]$. If j is not being continuously processed but preempted once or several times, the expression for C_j^{LP} in (7) is a lower bound on the real completion time. A more precise discussion of this matter is given in Lemma 10 a) below. Hence, (LP) is a relaxation of the scheduling problem under consideration.

A feasible solution y to (LP) does in general not correspond to a feasible preemptive schedule. Consider the following example:

Example 1. Let $J = \{1, \dots, n\}$ with $p_j = 1$ for all $j \in J$, $w_j = 0$ for $1 \leqslant j \leqslant n-1$, $w_n = 1$ and $j \prec n$ for $1 \leqslant j \leqslant n-1$. We get a feasible solution to (LP) if we set $y_{jt} = \frac{1}{n}$ for all $j \in J$ and $t = 0, \dots, T = n-1$.

In the solution to (LP) given in Example 1 several jobs share the capacity of the single machine in each time interval. Moreover, the precedence relations between job n and all other jobs are not obeyed since fractions of job n are processed on the machine before its predecessors are completed. However, the solution fulfills constraint (6) such that at any point in time the completed fraction of job n is not larger than the completed fraction of each of its predecessors.

Phillips et al. [29] introduced the term *fractional preemptive schedule* for a schedule in which a job can share the capacity of one machine with several other jobs. We extend this notion to the case with precedence constraints and call a fractional preemptive schedule P feasible if no job is processed before its release date and, at any point in time, the completed fraction of job j is not larger than the completed fraction of each of its predecessors. Carrying over the definition of $\boldsymbol{\alpha}$-points to fractional preemptive schedules, the latter condition is equivalent to $C_j^P(\alpha) \leqslant C_k^P(\alpha)$ for all $j \prec k$ and $0 \leqslant \alpha \leqslant 1$.

Corollary 3. *The results presented in Section 3, in particular Lemma 6, Corollary 1, and Lemma 9, still hold if P is a fractional preemptive schedule.*

Proof. The proofs of the above-mentioned results can directly be carried over to the more general setting of fractional preemptive schedules. □

We always identify a feasible solution to (LP) with the corresponding feasible fractional preemptive schedule and vice versa. We mutually use the interpretation that seems more suitable for our purposes. The expression for C_j^{LP} in (7) is called the *LP completion time* of job j.

The following lemma highlights the relation between the LP completion times and the α-points of jobs in the corresponding fractional preemptive schedule for a feasible solution to (LP). The observation in a) is due to Goemans [16]; it says that the LP completion time of job j is the sum of half its processing time and its mean busy time. An analogous result to b) was given in [22, Lemma 2.1] for a somewhat different LP.

Lemma 10. *Consider a feasible solution y to (LP) and the corresponding feasible fractional preemptive schedule P. Then, for each job j:*

a) $$C_j^{LP} = \frac{p_j}{2} + \frac{1}{p_j} \sum_{t=r_j}^{T} y_{jt}\big(t + \tfrac{1}{2}\big) = \frac{p_j}{2} + \int_0^1 C_j^P(\alpha)\, d\alpha \leqslant C_j^P$$

and equality holds if and only if $C_j^P = C_j^P(0) + p_j$;

b) $C_j^P(\alpha) \leqslant \dfrac{1}{1-\alpha} C_j^{LP}$ *for any constant $\alpha \in [0,1)$ and this bound is tight.*

Proof. For a job $j \in J$, we denote by ζ_j^t, $t = r_j, \dots, T+1$, the fraction of j that is finished in the fractional preemptive schedule P by time t. Since $0 = \zeta_j^{r_j} \leqslant \zeta_j^{r_j+1} \leqslant \dots \leqslant \zeta_j^{T+1} = 1$, we get

$$\begin{aligned}\int_0^1 C_j^P(\alpha)\,d\alpha &= \sum_{t=r_j}^{T} \int_{\zeta_j^t}^{\zeta_j^{t+1}} C_j^P(\alpha)\,d\alpha \\ &= \sum_{t=r_j}^{T} \left(\zeta_j^{t+1} - \zeta_j^t\right)\left(t + \tfrac{1}{2}\right)\end{aligned}$$

since $C_j^P(\alpha) = t + \frac{\alpha - \zeta_j^t}{\zeta_j^{t+1} - \zeta_j^t}$ for $\alpha \in (\zeta_j^t, \zeta_j^{t+1}]$,

$$= \sum_{t=r_j}^{T} \frac{y_{jt}}{p_j}\left(t + \tfrac{1}{2}\right) \ .$$

The machine capacity constraints (5) yield $C_j^P(\alpha) \leqslant C_j^P - (1-\alpha)\,p_j$ for $\alpha \in [0,1]$, thus

$$\int_0^1 C_j^P(\alpha)\,d\alpha \ \leqslant\ C_j^P - p_j \int_0^1 (1-\alpha)\,d\alpha \ =\ C_j^P - \frac{p_j}{2} \ .$$

Equality holds if and only if $C_j^P(\alpha) = C_j^P - (1-\alpha)\,p_j$ for $0 \leqslant \alpha \leqslant 1$, which is equivalent to $C_j^P(0) = C_j^P - p_j$. This completes the proof of a). As a consequence we get for $\alpha \in [0,1]$

$$(1-\alpha)\,C_j^P(\alpha) \ \leqslant\ \int_\alpha^1 C_j^P(x)\,dx \ \leqslant\ \int_0^1 C_j^P(x)\,dx \ \leqslant\ C_j^{LP} \ .$$

In order to prove the tightness of this bound, we consider a job j with $p_j = 1$, $r_j = 0$ and an LP solution satisfying $y_{j0} = \alpha - \epsilon$ and $y_{jT} = 1 - \alpha + \epsilon$, where $\epsilon > 0$ small. This yields $C_j^{LP} = 1 + (1 - \alpha + \epsilon)\,T$ and $C_j^P(\alpha) \geqslant T$. Thus, for ϵ arbitrarily small and T arbitrarily large, the given bound gets tight. □

As a result of Lemma 10 b) the value of the fractional preemptive schedule given by a solution to (LP) can be arbitrarily bad compared to its LP value. Nevertheless, one can use the information that is contained in the structure of an LP solution in order to construct a feasible schedule whose value can be bounded in terms of the LP value, as will be shown in the following sections.

Notice that we cannot solve (LP) in polynomial time, but only in pseudo-polynomial time, since T and therefore the number of variables y_{jt} is only pseudo-polynomial in the input size of the problem. Schulz and Skutella [37] describe a closely related and only slightly worse LP relaxation of polynomial size in the general context of unrelated parallel machines. The idea is to change to new variables which are not associated with exponentially many time intervals of length 1, but rather with a polynomial number of intervals of geometrically increasing size. We omit the technical details in this survey.

In the absence of precedence constraints, it was indicated by Dyer and Wolsey [13] that (LP) can be solved in $O(n \log n)$ time. Goemans [15] developed the following result (see also [17]).

Theorem 6. *For instances of the problems* $1|\, r_j \,(,\, pmtn) \,|\sum w_j C_j$*, the linear programming relaxation* (LP) *can be solved in* $O(n \log n)$ *time and the preemptive list schedule in order of non-decreasing ratios* p_j/w_j *corresponds to an optimal solution.*

Proof. In the absence of precedence relations, constraints (6) are missing in (LP). Moreover, we can eliminate the variables C_j^{LP} from the relaxation by plugging (7) into the objective function. What remains is a transportation problem and, as a result, y_{jt} can be assumed to be integral.

The remaining part of the proof is based on an interchange argument: Consider any optimal $0/1$-solution y to (LP) and suppose that $j < k$ (such that $p_j/w_j \leqslant p_k/w_k$), $r_j \leqslant t < \tau$, and $y_{kt} = y_{j\tau} = 1$. Then, replacing $y_{kt} = y_{j\tau}$ by 0 and $y_{k\tau} = y_{jt}$ by 1 gives another feasible solution to (LP) with an increase in the objective function of $(\tau - t)\left(\frac{w_k}{p_k} - \frac{w_j}{p_j}\right) \leqslant 0$. Thus, the new solution is also optimal and through iterative application of this interchange argument we arrive at the solution that corresponds to the preemptive list schedule in order of non-decreasing ratios p_j/w_j. □

It follows from Theorem 6 that in the absence of precedence constraints and release dates an optimal single machine schedule corresponds to an optimal solution to (LP). Moreover, if we allow release dates and preemption, the optimal solution to (LP) described in Theorem 6 is a feasible schedule that minimizes the weighted sum of mean busy times. It is also shown in [17] that, in the absence of precedence constraints, the LP relaxation (LP) is equivalent to an LP in completion time variables which defines a supermodular polyhedron.

5.1 Another Time-Indexed LP Relaxation

For the non-preemptive problem $1|\, r_j \,|\sum w_j C_j$ Dyer and Wolsey [13] also proposed another time-indexed LP relaxation that can easily be extended to the setting with precedence constraints; see Figure 3. Again, for each job j and each integral point in time $t = 0, \dots, T$, we introduce a decision variable x_{jt}. However, now the variable is set to 1 if j *starts* processing at time t and to 0 otherwise. Note that the x_{jt} variables do not have the preemptive flavor of the y_{jt} variables and therefore lead to an LP relaxation for non-preemptive single machine scheduling only.

In the absence of precedence constraints, Dyer and Wolsey showed that this relaxation is stronger than (LP). In fact, even for instances with precedence constraints, every feasible solution to (LP') can easily be transformed into a solution to (LP) of the same value by assigning

$$y_{jt} := \sum_{\ell=\max\{0,t-p_j+1\}}^{t} x_{j\ell} \qquad \text{for } j \in J \text{ and } t = 0, \dots, T.$$

In particular, we can interpret a feasible solution to (LP') as a feasible fractional preemptive schedule and the results in Lemma 10 also hold in this case.

$$
(LP')\qquad
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{j\in J} w_j C_j^{LP} & \\
\text{subject to} & \displaystyle\sum_{t=r_j}^{T} x_{jt} = 1 & \text{for all } j \in J, \\
& \displaystyle\sum_{j\in J}\ \sum_{\ell=\max\{0,t-p_j+1\}}^{t} x_{j\ell} \leqslant 1 & \text{for all } t = 0,\dots,T, \qquad (8)\\
& \displaystyle\sum_{\ell=r_j}^{t} x_{j\ell} \geqslant \sum_{\ell=r_k}^{t+p_j} x_{k\ell} & \text{for all } j \prec k \text{ and } t = r_j,\dots,T-p_j, \qquad (9)\\
& \displaystyle C_j^{LP} = p_j + \sum_{t=r_j}^{T} t\, x_{jt} & \text{for all } j \in J, \\
& x_{jt} = 0 & \text{for all } j \in J \text{ and } t = 0,\dots,r_j-1, \qquad (10)\\
& x_{jt} \geqslant 0 & \text{for all } j \in J \text{ and } t = r_j,\dots,T.
\end{array}
$$

Fig. 3. The time-indexed LP relaxation (LP')

5.2 Combinatorial Relaxations

In this subsection we show that the LP relaxation (LP') is equivalent to a combinatorial relaxation of $1|\, r_j,\, prec \,|\sum w_j C_j$. We interpret an instance of this scheduling problem as a game for one person who is given the set of jobs J and the single machine and wants to find a feasible schedule of minimum value.

We consider the following relaxation of this game: Assume that there are k players $1,\dots,k$ instead of only one. Each player i is given one machine and a set of jobs $J_i = J$. Moreover, the players are allowed to cooperate by exchanging jobs. To be more precise, a job of player i can be scheduled on an arbitrary machine rather than only on i's machine. However, each player has to respect release dates and precedence constraints of his jobs. The aim is to minimize the average over all players of the weighted sum of completion times of their jobs. This relaxation is called *k-player relaxation*, a feasible solution is a *k-player schedule*. Moreover, if k is not fixed but can be chosen, we call the resulting relaxation *multi-player relaxation* and a feasible solution is a *multi-player schedule*.

As an example, consider the following single machine instance without precedence constraints consisting of four jobs:

job j	r_j	p_j	w_j
1	6	2	6
2	0	3	3
3	0	2	2
4	0	2	2

Note that job 1 is the most important job. The instance is constructed such that either at least one unit of idle time has to be introduced before the start of job 1 or this job has to be delayed until time 7. Both alternatives are optimal and give a schedule with value 87. However, there exists a 2-player schedule of value 83.5: Both players start their first job at time 6. Player 1 starts his second job at time 0 and then, at time 3, the second job of player 2 on his machine. Player 2 processes the four remaining jobs of length 2 in the intervals $(0, 6]$ and $(8, 10]$ on his machine. Notice that the two players neither delay their first jobs nor introduce idle time before their start. As a result of this example we get the following corollary.

Corollary 4. *The 2-player relaxation is not better than a $\frac{174}{167}$-relaxation for the problem $1|\,r_j\,|\sum w_j C_j$.*

The negative result in Corollary 4 can be slightly improved to a bound of $1 + 1/(2\sqrt{55}+9) > \frac{174}{167}$ by increasing the processing time of job 1 to $(\sqrt{55}-3)/2$ and its weight to $\sqrt{55}-1$. The main result of this subsection is the following theorem:

Theorem 7. *The LP relaxation (LP') of the problem $1|\,r_j,\, prec\,|\sum w_j C_j$ is equivalent to the multi-player relaxation.*

Proof. We first argue that each k-player schedule corresponds to a feasible solution of (LP') with the same value: For all $j \in J$ and $t = 0, \ldots, T$ let x_{jt} be the number of players whose job $j \in J_i$ is started at time t divided by k. It follows from the definition of the relaxation that x satisfies all constraints of (LP') and is thus a feasible solution. To understand that constraints (9) are fulfilled, observe that each player has to respect the precedence constraints of his jobs. Moreover, the value of x is equal to the value of the k-player schedule by definition of x.

On the other hand, we can show that for each feasible solution x whose values x_{jt} are all rational numbers there exists a k such that x corresponds to a feasible solution of the k-player relaxation. Let k be the least common multiple of the denominators of the rational values x_{jt}. For each job $j \in J$ and each player $i = 1, \ldots, k$, let t_{ji} the smallest integer such that $\sum_{\ell=0}^{t_{ji}} x_{jt} \geqslant \frac{i}{m}$. The value t_{ji} is the starting time of i's job j in the k-player schedule that we construct by induction over $t = 0, \ldots, T$: At time $t = 0$ start all jobs $j \in J$ of players i with $t_{ji} = 0$ on the k machines. Notice that the number of these job-player pairs is bounded by k since $\sum_{j \in J} x_{j0} \leqslant 1$ by constraints (8). Assume that for all players i all jobs j with $t_{ji} < t$ have been started at time t_{ji}. Then, the number of idle machines at time t is equal to

$$k\left(1 - \sum_{j \in J} \sum_{\ell=\max\{0, t-p_j+1\}}^{t-1} x_{j\ell}\right) .$$

Therefore, by constraints (8), there are sufficiently many idle machines at time t such that all jobs j of players i with $t_{ji} = t$ can be started. This k-player schedule respects release dates by constraints (10). Moreover, for a pair of jobs $j \prec k$ and each player i we get $t_{ji} + p_j \leqslant t_{ki}$ by the definition of the t_{ji}'s and constraints (9). $\square$

Similar interpretations of time-indexed LP relaxations have been given by van den Akker [2, Proof of Theorem 3.1] and Schulz [33]. Chan, Muriel, and Simchi-Levi [8] used this idea to prove results on the quality of those time-indexed LP relaxations.

It is an interesting direction for future research to investigate how the structure of an optimal k-player schedule can be used to construct provably good schedules. Another interesting question is if one can restrict to special values of k in order to get an optimal multi-player schedule, e. g., $k \leqslant n$ or $k \in O(n)$, or values of k depending on the given instance. This would also yield results on the structure of the polyhedron corresponding to (LP'), i. e., results on an optimal vertex or on general vertices (like $\frac{1}{n}$-integrality etc.).

Another interesting topic is the complexity of the k-player relaxation for special values of k. Of course, it is NP-hard for $k = 1$. We conjecture that it is NP-hard to find an optimal k-player schedule for each fixed k. We also guess that it is NP-hard to compute an optimal multi-player schedule when k is not fixed but bounded by the number of jobs n.

5.3 On the Quality of Time-Indexed LP Relaxations

We conclude this section with negative and positive results on the quality of the relaxations (LP) and (LP'). The positive results follow from the LP-based approximations discussed in the following sections. The lower bound for (LP) and $1|\, r_j \,|\sum w_j C_j$ is due to Goemans et al. [17]. A summary of the results is given in Table 2.

Theorem 8.

a) The LP relaxations (LP) and (LP') are 2-relaxations for the scheduling problem $1|\, prec \,|\sum w_j C_j$ and this bound is tight.
b) (LP) is a 2-relaxation for the problem $1|\, r_j,\, prec,\, pmtn \,|\sum w_j C_j$ and this bound is tight.
c) (LP) and (LP') are e-relaxations for the problem $1|\, r_j,\, prec \,|\sum w_j C_j$ and not better than 2-relaxations.
d) (LP) is a $\frac{4}{3}$-relaxation and not better than an $\frac{8}{7}$-relaxation for the scheduling problem $1|\, r_j,\, pmtn \,|\sum w_j C_j$.
e) (LP) is a 1.6853-relaxation and not better than an $\frac{e}{e-1}$-relaxation for the problem $1|\, r_j \,|\sum w_j C_j$.
f) (LP') is a 1.6853-relaxation and not better than a $\frac{174}{167}$-relaxation for the problem $1|\, r_j \,|\sum w_j C_j$ $\left(\frac{174}{167} \approx 1.0419\right)$.

Proof. It is shown in Section 8 that (LP) is a 2-relaxation for $1|\, prec \,|\sum w_j C_j$ and $1|\, r_j,\, prec,\, pmtn \,|\sum w_j C_j$, and an e-relaxation for the problem $1|\, r_j,\, prec \,|\sum w_j C_j$. Moreover, as mentioned above, (LP') is a stronger relaxation than (LP) for the non-preemptive problems. To prove the negative results in a), b), and c) we use the instance given in Example 1 and consider the feasible solutions to (LP) and (LP') given by $y_{jt} = x_{jt} = \frac{1}{n}$ for $j = 1, \dots, n-1$ and $t = 0, \dots, n-1$; moreover, we set $y_{nt} = x_{nt} = \frac{1}{n}$ for $t = 1, \dots, n$. The value of an optimal schedule is n whereas the value of the given solutions to (LP) and (LP') is $\frac{n+3}{2}$. When n goes to infinity, the ratio of these two values converges to 2.

Table 2. Summary of results on the quality of the time-indexed LP relaxations (LP) and (LP') given in Theorem 8

Results on the quality of (LP) and (LP')				
problem	(LP)		(LP')	
	lower bound	upper bound	lower bound	upper bound
$1\|\, r_j \,\|\sum w_j C_j$	1.5820	1.6853	1.0419	1.6853
$1\|\, r_j,\, pmtn \,\|\sum w_j C_j$	$\frac{8}{7}$	$\frac{4}{3}$	—	—
$1\|\, prec \,\|\sum w_j C_j$	2	2	2	2
$1\|\, r_j,\, prec \,\|\sum w_j C_j$	2	e	2	e
$1\|\, r_j,\, prec,\, pmtn \,\|\sum w_j C_j$	2	2	—	—

The positive results in d), e), and f) are derived in Section 7. To prove the negative result in d), consider the following instance with n jobs where n is assumed to be even. The processing times of the first $n-1$ jobs $j = 1, \ldots, n-1$ are 1, their common release date is $\frac{n}{2}$, and all weights are $\frac{1}{n^2}$. The last job has processing time $p_n = n$, weight $w_n = \frac{1}{2n}$, and is released at time 0. This instance is constructed such that every reasonable preemptive schedule without idle time on the machine has value $2 - \frac{3}{2n}$. However, an optimal solution to (LP) given in Theorem 6 has value $\frac{7}{4} - \frac{5}{4n}$ such that the ratio goes to $\frac{8}{7}$ when n gets large.

We use Theorem 5 in order to prove the negative result in e). There we have argued that the ratio between the value of an optimal non-preemptive schedule and the value of an optimal preemptive schedule can be arbitrarily close to $\frac{e}{e-1}$. The optimal LP value is a lower bound on the value of an optimal preemptive schedule; this completes the proof of e).

The negative result in f) follows from Corollary 4 and Theorem 7. □

It is an open problem and an interesting direction for future research to close the gaps between the lower and the upper bounds highlighted in Table 2. We conjecture that the lower bound of $\frac{e}{e-1}$ for the relaxation (LP) of the problem $1|\, r_j \,|\sum w_j C_j$ is not tight.

For the relaxation (LP') of the same problem we strongly believe that the precise ratio is closer to the lower than to the upper bound. We hope that the combinatorial interpretation given in Subsection 5.2 will lead to improved approximation results and upper bounds on the quality of the relaxation (LP').

Remark 1. For the problem with precedence constraints the relaxation (LP) can be slightly strengthened by replacing (6) with the stronger constraints

$$\frac{1}{p_j}\sum_{\ell=r_j}^{t} y_{j\ell} \geqslant \frac{1}{p_k}\sum_{\ell=r_k}^{t+\min\{p_j,p_k\}} y_{k\ell} \quad \text{for all } j \prec k \text{ and } t = 0, \ldots, T - \min\{p_j, p_k\} \ .$$

However, the relaxation is not stronger than (LP') by constraints (9), and therefore not better than a 2-relaxation.

Potts [30] introduced a linear programming relaxation of $1|\,prec\,|\sum w_j C_j$ in linear ordering variables. Hall et al. [21] showed that this relaxation is a 2-relaxation. Chekuri and Motwani [10] provided a class of instances showing that this result is tight.

6 Scheduling in Order of LP Completion Times

Schulz [34] (see also [21]) introduced the idea of LIST SCHEDULING in order of LP completion times and applied it to get approximation algorithms for quite a few problems to minimize the weighted sum of completion times. In this section we briefly review his results for the scheduling problems under consideration in order to give the reader the opportunity to compare the underlying ideas and intuition with the concept of α-points.

Schulz used a strengthened version of an LP relaxation in completion time variables that was introduced by Wolsey [52] and Queyranne [31]. However, the technique and analysis can also be applied to the stronger relaxation (LP) in time-indexed variables. The following lemma contains the key insight and is a slightly weaker version of [34, Lemma 1].

Lemma 11. *Consider a feasible solution y to (LP). Then, for each job $j \in J$*

$$\sum_{\substack{k \\ C_k^{LP} \leqslant C_j^{LP}}} p_k \leqslant 2\, C_j^{LP} \ .$$

Proof. Let $K := \{k \in J \mid C_k^{LP} \leqslant C_j^{LP}\}$ and $p(K) := \sum_{k \in K} p_k$. This yields

$$\begin{aligned} p(K)\, C_j^{LP} &\geqslant \sum_{k \in K} p_k\, C_k^{LP} \\ &\geqslant \sum_{k \in K} \sum_{t=0}^{T} y_{kt} \left(t + \tfrac{1}{2}\right) && \text{by Lemma 10 a)} \\ &= \sum_{t=0}^{T} \left(t + \tfrac{1}{2}\right) \sum_{k \in K} y_{kt} \ ; \end{aligned}$$

using the constraints (5) and (4) the last term can be bounded by

$$\geqslant \sum_{t=0}^{p(K)-1} \left(t + \tfrac{1}{2}\right) = \tfrac{1}{2}\, p(K)^2 \ .$$

Dividing the resulting inequality by $p(K)$ yields the result. □

With Lemma 11 at hand one can prove the following theorem:

Theorem 9. *Given a feasible solution to (LP),* LIST SCHEDULING *in order of non-decreasing LP completion times yields a feasible schedule where the completion time of each job j is bounded by $2\, C_j^{LP}$ in the absence of nontrivial release dates, and by $3\, C_j^{LP}$ else.*

Proof. We denote the fractional preemptive schedule corresponding to the given feasible LP solution by P. To check the feasibility of the computed schedule observe that $j \prec k$ implies $C_j^P(\alpha) \leqslant C_k^P(\alpha)$ for $0 \leqslant \alpha \leqslant 1$, and thus $C_j^{LP} \leqslant C_k^{LP}$ by Lemma 10 a). Therefore, if ties are broken in accordance with the precedence constraints the sequence is feasible.

In the absence of nontrivial release dates, the completion time of each job j is the sum of the processing times of jobs that start no later than j. Thus, the bound of 2 is a direct consequence of Lemma 11.

Since $\max_{k: C_k^{LP} \leqslant C_j^{LP}} r_k$ is a lower bound on C_j^{LP}, the bound of 3 follows from Lemma 11 and Lemma 3. □

The LP relaxation in completion time variables that is used in [34] can be solved in polynomial time. Therefore, computing an optimal solution to the LP relaxation and then applying LIST SCHEDULING in order of non-decreasing LP completion times is a 2-approximation algorithm for $1|\, prec \,|\sum w_j C_j$ and a 3-approximation algorithm for $1|\, r_j,\, prec \,|\sum w_j C_j$. For the problem without nontrivial release dates, Schulz proves a slightly better performance guarantee of $2 - \frac{2}{n+1}$.

Hall et al. [21] show that PREEMPTIVE LIST SCHEDULING in order of non-decreasing LP completion times for an appropriate LP relaxation in completion time variables is a 2-approximation algorithm for the problem $1|\, r_j,\, prec,\, pmtn \,|\sum w_j C_j$.

Möhring, Schulz, and Uetz [27] study the problem of minimizing the total weighted completion time in stochastic machine scheduling. Job processing times are not known in advance, they are realized on-line according to given probability distributions. The aim is to find a scheduling policy that minimizes the average weighted completion time in expectation. They generalize results from deterministic scheduling and derive constant-factor performance guarantees for priority rules which are guided by optimal LP solutions in completion time variables. Skutella and Uetz [43, 44] generalize this approach and give approximation algorithms with constant performance guarantee for precedence-constrained scheduling problems.

7 Approximations for Single Machine Scheduling with Release Dates

In this section we present approximation algorithms for the problems $1|\, r_j \,|\sum w_j C_j$ and its preemptive variant $1|\, r_j,\, pmtn \,|\sum w_j C_j$ that have been obtained by Goemans et al. [17] and Schulz and Skutella [36], respectively. The first constant-factor approximation algorithms to minimize the total weighted completion time on a single machine subject to release dates are due to Phillips et al. [29]. They consider an LP relaxation that is also based on time-indexed variables which have a different meaning however. Based on an optimal solution to this relaxation they apply an idea which is somehow related to PREEMPTIVE LIST SCHEDULING in order of α-points for $\alpha = \frac{1}{2}$. This leads to an approximation algorithm with performance guarantee $8 + \varepsilon$ for the generalization of the problem $1|\, r_j,\, pmtn \,|\sum w_j C_j$ to the setting of unrelated parallel machines. Together with the conversion technique described in Theorem 2 this yields a $(16 + \varepsilon)$-approximation algorithm for $1|\, r_j \,|\sum w_j C_j$.

Hall et al. [22] use LIST SCHEDULING in order of α-points for several problems and different but fixed values of α based on the LP relaxation (LP') in order to compute provably good schedules. However, their definition of α-points slightly differs from the one discussed here. Based on a different approach that relies on the results of Shmoys and Tardos [40] for the generalized assignment problem, they give a 4-approximation algorithm for $1|\,r_j\,|\sum w_jC_j$, which has subsequently been improved to performance guarantee 3, see Section 6 and [21].

Independently from each other, Chekuri et al. [11] (see Section 4) and Goemans [16] have taken up the idea of converting preemptive schedules into non-preemptive ones by LIST SCHEDULING in order of α-points (as introduced in [29, 22]), and enriched it by the use of randomness. Recently, Afrati et al. [1] gave polynomial-time approximation schemes for the problems considered in this section and generalizations thereof.

We analyze the following simple randomized algorithm for single machine scheduling with release dates. We consider both the non-preemptive and the preemptive variant of the algorithm which only differ in the last step:

```
Algorithm: RANDOM-$\boldsymbol{\alpha}$
  1) Construct the preemptive list schedule $P$ in order of non-decreasing ratios
     $p_j/w_j$.
  2) For each job $j \in J$, draw $\alpha_j$ randomly from $[0,1]$.
  3) Output the resulting (preemptive) $\boldsymbol{\alpha}$-schedule.
```

Since (PREEMPTIVE) LIST SCHEDULING can be implemented to run in $O(n \log n)$ time by Lemma 1, the running time of Algorithm RANDOM-$\boldsymbol{\alpha}$ is $O(n \log n)$. It follows from Theorem 6 that the first step of Algorithm RANDOM-$\boldsymbol{\alpha}$ implicitly computes an optimal solution to the relaxation (LP). This observation is used in the analysis. Note, however, that the algorithm is purely combinatorial and can be formulated and implemented without even knowing the LP relaxation.

While the total number of possible orderings of jobs is $n! = 2^{O(n \log n)}$, it is shown in [17] that the maximal number of different $\boldsymbol{\alpha}$-schedules which can be computed by Algorithm RANDOM-$\boldsymbol{\alpha}$ is at most 2^{n-1} and this bound can be attained. In particular, Algorithm RANDOM-$\boldsymbol{\alpha}$ could also be formulated over a discrete probability space. Due to the possibly exponential number of different $\boldsymbol{\alpha}$-schedules, we cannot afford to derandomize Algorithm RANDOM-$\boldsymbol{\alpha}$ by enumerating all (α_j)-schedules. One can instead use the method of conditional probabilities [28] and the derandomized version can be implemented to run in $O(n^2)$ time; we refer to [17] for details.

As in Section 4, the variant of Algorithm RANDOM-$\boldsymbol{\alpha}$ with only one random variable α for all jobs instead of individual α_j's is of special interest. We denote this variant by RANDOM-α and it follows from Proposition 1 that it can be derandomized yielding a deterministic algorithm with running time $O(n^2)$. This deterministic algorithm is called BEST-α. The proof of the following results on the performance of Algorithm RANDOM-α (and thus of Algorithm BEST-α) can be found in [17].

Theorem 10.

a) *For fixed α the performance guarantee of Algorithm* RANDOM-α *is* $\max\{1+\frac{1}{\alpha}, 1+2\alpha\}$. *In particular, for $\alpha = 1/\sqrt{2}$ the performance guarantee is* $1+\sqrt{2}$.

b) If α is drawn uniformly at random from $[0,1]$, then the expected performance guarantee of RANDOM-α *is* 2.

The same performance ratio of 2 is also achieved by Algorithm RANDOM-$\boldsymbol{\alpha}$ with independent and uniformly distributed random variables α_j. However, in this case the analysis is noticeably simpler and more general. In particular it does not make use of the special structure of the preemptive list schedule P but works for an arbitrary fractional preemptive schedule P corresponding to a feasible solution to (LP); see Lemma 12 below.

Theorem 11. *Let the random variables α_j be pairwise independently and uniformly drawn from $[0,1]$. Then, Algorithm* RANDOM-$\boldsymbol{\alpha}$ *achieves expected performance guarantee 2 for the scheduling problems $1|\,r_j\,|\sum w_jC_j$ and $1|\,r_j,\,pmtn\,|\sum w_jC_j$.*

Theorem 11 is a consequence of the stronger result in Lemma 12. Starting with an arbitrary fractional preemptive schedule in the first step of RANDOM-$\boldsymbol{\alpha}$ we can relate the value of the schedule computed by the algorithm to the corresponding LP value.

Lemma 12. *Let y be a feasible solution to (LP) and denote the corresponding fractional preemptive schedule by P. Suppose that the random variables α_j are pairwise independently and uniformly drawn from $[0,1]$. Then, for each job $j \in J$, its expected completion time in the corresponding $\boldsymbol{\alpha}$-schedule is at most $2\,C_j^{LP}$.*

Proof. We consider an arbitrary, but fixed job $j \in J$. To analyze the expected completion time of j, we first keep α_j fixed, and consider the conditional expectation $\mathrm{E}_{\alpha_j}[C_j^{\boldsymbol{\alpha}}]$. Since the random variables α_j and α_k are independent for each $k \neq j$, Corollary 1 and Lemma 6 yield

$$\begin{aligned}
\mathrm{E}_{\alpha_j}[C_j^{\boldsymbol{\alpha}}] &\leqslant C_j^P(\alpha_j) + p_j + \sum_{k\neq j} p_k \int_0^{\eta_k(\alpha_j)} \big(1+\alpha_k-\eta_k(\alpha_j)\big)\,d\alpha_k \\
&= C_j^P(\alpha_j) + p_j + \sum_{k\neq j} p_k \Big(\eta_k(\alpha_j) - \frac{\eta_k(\alpha_j)^2}{2}\Big) \\
&\leqslant C_j^P(\alpha_j) + p_j + \sum_{k\neq j} p_k\,\eta_k(\alpha_j) \;\leqslant\; 2\big(C_j^P(\alpha_j) + \tfrac{1}{2}p_j\big) \;.
\end{aligned}$$

To get the unconditional expectation $\mathrm{E}[C_j^{\boldsymbol{\alpha}}]$ we integrate over all possible choices of α_j:

$$\mathrm{E}[C_j^{\boldsymbol{\alpha}}] = \int_0^1 \mathrm{E}_{\alpha_j}[C_j^{\boldsymbol{\alpha}}]\,d\alpha_j \;\leqslant\; 2\Big(\int_0^1 C_j^P(\alpha_j)\,d\alpha_j + \frac{p_j}{2}\Big) = 2\,C_j^{LP} \;;$$

the last equation follows from Lemma 10 a). □

Proof (of Theorem 11). Algorithm RANDOM-$\boldsymbol{\alpha}$ starts with an optimal solution to (LP) whose value is a lower bound on the value of an optimal (preemptive) schedule. We compare the expected value of the (preemptive) $\boldsymbol{\alpha}$-schedule to this lower bound. Using Lemma 12 and linearity of expectations we get

$$\mathrm{E}\Big[\sum_j w_j C_j^{\boldsymbol{\alpha}}\Big] \;=\; \sum_j w_j \mathrm{E}[C_j^{\boldsymbol{\alpha}}] \;\leqslant\; 2\sum_j w_j C_j^{LP} \;\;.$$

Since for each fixed $\boldsymbol{\alpha}$ and each job j the completion time of j in the preemptive $\boldsymbol{\alpha}$-schedule is always less than or equal to its completion time in the non-preemptive $\boldsymbol{\alpha}$-schedule, the preemptive variant of Theorem 11 follows from the result for the non-preemptive case. We have even shown that the non-preemptive variant of Algorithm RANDOM-$\boldsymbol{\alpha}$ computes a schedule whose expected value is bounded by twice the value of an optimal preemptive schedule. Thus, even this algorithm constitutes a randomized 2-approximation algorithm for $1|\,r_j,\,pmtn\,|\sum w_j C_j$. □

The expected performance guarantee of Algorithm RANDOM-$\boldsymbol{\alpha}$ can be improved beyond 2 by using more intricate density functions and by exploiting the special structure of the preemptive list schedule P in the first step of Algorithm RANDOM-$\boldsymbol{\alpha}$.

We start with an analysis of the structure of the preemptive list schedule P. Consider any job j, and assume that, in the preemptive schedule P, job j is preempted at time s and its processing resumes at time $t > s$. Then all jobs which are processed between s and t have a smaller index; as a result, these jobs will be completely processed between times s and t. Thus, in the preemptive list schedule P, between the start time and the completion time of any job j, the machine is constantly busy, alternating between the processing of portions of j and the complete processing of groups of jobs with smaller index. Conversely, any job preempted at the start time $C_j^P(0)$ of job j will have to wait at least until job j is complete before its processing can be resumed.

We capture this structure by partitioning, for a fixed job j, the set of jobs $J \setminus \{j\}$ into two subsets J_1 and J_2: Let J_2 denote the set of all jobs that are processed between the start and completion of job j. All remaining jobs are put into subset J_1. Notice that the function η_k is constant for jobs $k \in J_1$; to simplify notation we write $\eta_k := \eta_k(\alpha_j)$ for those jobs. The same holds for the function t_{idle} since no idle time occurs between the start and the completion of job j in P; we thus write t_{idle} instead of $t_{\mathrm{idle}}(\alpha_j)$. For $k \in J_2$, let $0 < \mu_k < 1$ denote the fraction of job j that is processed before the start of job k; the function η_k is then given by

$$\eta_k(\alpha_j) \;=\; \begin{cases} 0 & \text{if } \alpha_j \leqslant \mu_k, \\ 1 & \text{if } \alpha_j > \mu_k, \end{cases} \qquad\qquad \text{for } k \in J_2.$$

We can now rewrite the equation in Lemma 6 as

$$C_j^P(\alpha_j) \;=\; t_{\mathrm{idle}} + \sum_{k\in J_1} \eta_k p_k \;+ \sum_{\substack{k\in J_2\\ \alpha_j > \mu_k}} p_k \;+\; \alpha_j\, p_j \;=\; C_j^P(0) \;+ \sum_{\substack{k\in J_2\\ \alpha_j > \mu_k}} p_k \;+\; \alpha_j\, p_j \;\;. \tag{11}$$

Plugging (11) into Lemma 10 a) yields

$$C_j^{LP} \;=\; C_j^P(0) \;+\; \sum_{k\in J_2} (1-\mu_k) p_k \;+\; p_j \;\;. \tag{12}$$

Moreover, Corollary 1 can be rewritten as

$$C_j^{\boldsymbol{\alpha}} \leqslant C_j^P(0) + \sum_{\substack{k \in J_1 \\ \alpha_k \leqslant \eta_k}} (1 + \alpha_k - \eta_k)\, p_k + \sum_{\substack{k \in J_2 \\ \alpha_j > \mu_k}} (1 + \alpha_k)\, p_k + (1 + \alpha_j)\, p_j \ , \tag{13}$$

where, for $k \in J_2$, we have used the fact that $\alpha_k \leq \eta_k(\alpha_j)$ is equivalent to $\alpha_j > \mu_k$. Similarly, Lemma 8 can be rewritten as

$$C_j^{\boldsymbol{\alpha}-pmtn} \leqslant C_j^P(0) + \sum_{\substack{k \in J_1 \\ \alpha_k \leqslant \eta_k}} (1 - \eta_k)\, p_k + \sum_{\substack{k \in J_2 \\ \alpha_j > \mu_k}} p_k + p_j \ . \tag{14}$$

The expressions (11), (12), (13), and (14) reflect the structural insights that we need for proving stronger bounds for $\boldsymbol{\alpha}$-schedules and preemptive $\boldsymbol{\alpha}$-schedules in the sequel.

As mentioned above, the second ingredient for an improvement on the bound of 2 in Theorem 11 is a more sophisticated probability distribution of the random variables α_j. In view of the bound on $C_j^{\boldsymbol{\alpha}}$ given in (13), we have to cope with two contrary phenomena: On the one hand, small values of α_k keep the terms of the form $(1 + \alpha_k - \eta_k)$ and $(1+\alpha_k)$ on the right-hand side of (13) small; on the other hand, choosing larger values decreases the number of terms in the first sum on the right-hand side of (13). The balancing of these two effects contributes to reducing the bound on the expected value of $C_j^{\boldsymbol{\alpha}}$. Similar considerations can be made for the preemptive case and the bound given in (14).

7.1 A 1.6853-Approximation Algorithm for $1|\, r_j \,|\sum w_j C_j$

In this subsection, we will prove the following theorem that was achieved by Goemans et al. [17].

Theorem 12. *Let* $\gamma \approx 0.4835$ *be the unique solution to the equation*

$$\gamma + \ln(2 - \gamma) = e^{-\gamma}\big((2 - \gamma)e^{\gamma} - 1\big)$$

satisfying $0 < \gamma < 1$. *Define* $\delta := \gamma + \ln(2 - \gamma) \approx 0.8999$ *and* $c := 1 + e^{-\gamma}/\delta < 1.6853$. *Let the* α_j*'s be chosen pairwise independently from a probability distribution over* $[0, 1]$ *with the density function*

$$f(\alpha) = \begin{cases} (c-1)e^{\alpha} & \text{if } \alpha \leqslant \delta, \\ 0 & \text{otherwise,} \end{cases}$$

see Figure 4. Then, the expected completion time of every job j *in the non-preemptive schedule constructed by Algorithm* RANDOM-$\boldsymbol{\alpha}$ *is at most* $c\,C_j^{LP}$ *and the expected performance guarantee of Algorithm* RANDOM-$\boldsymbol{\alpha}$ *is* $c < 1.6853$.

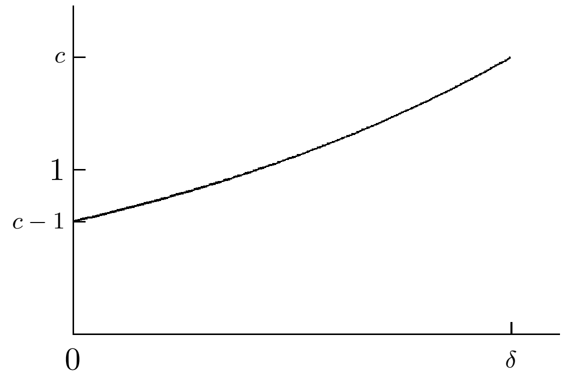

Fig. 4. The density function used for $1|\, r_j \,|\sum w_j C_j$ in Theorem 12

The bound in Theorem 12 yields also a bound on the quality of the relaxations (LP):

Corollary 5. *The relaxation* (LP) *is a* 1.6853*-relaxation of* $1|\, r_j \,|\sum w_j C_j$.

Before we prove Theorem 12 we state two properties of the density function f that are crucial for the analysis of the corresponding random $\boldsymbol{\alpha}$-schedule.

Lemma 13. *The function f given in Theorem 12 is a density function with the following properties:*

(i) $\displaystyle\int_0^\eta f(\alpha)(1+\alpha-\eta)\,d\alpha \;\leqslant\; (c-1)\,\eta$ *for all* $\eta \in [0,1]$,

(ii) $\displaystyle(1+\mathrm{E}_f)\int_\mu^1 f(\alpha)\,d\alpha \;\leqslant\; c\,(1-\mu)$ *for all* $\mu \in [0,1]$,

where E_f *denotes the expected value of a random variable* α *that is distributed according to* f.

Property (i) is used to bound the delay to job j caused by jobs in J_1 which corresponds to the first summation on the right-hand side of (13). The second summation reflects the delay to job j caused by jobs in J_2 and will be bounded by property (ii).

Proof (of Lemma 13*).* A short computation shows that $\delta = \ln\frac{c}{c-1}$. The function f is a density function since

$$\int_0^1 f(\alpha)\,d\alpha \;=\; (c-1)\int_0^\delta e^\alpha\,d\alpha \;=\; (c-1)\Big(\frac{c}{c-1}-1\Big) \;=\; 1\ .$$

In order to prove property (i), observe that for $\eta \in [0,\delta]$

$$\int_0^\eta f(\alpha)(1+\alpha-\eta)\,d\alpha \;=\; (c-1)\int_0^\eta e^\alpha(1+\alpha-\eta)\,d\alpha \;=\; (c-1)\,\eta\ .$$

For $\eta \in (\delta, 1]$ we therefore get

$$\int_0^\eta f(\alpha)(1+\alpha-\eta)\,d\alpha \;<\; \int_0^\delta f(\alpha)(1+\alpha-\delta)\,d\alpha \;=\; (c-1)\,\delta \;<\; (c-1)\,\eta \;.$$

In order to prove property (ii), we first compute

$$\mathrm{E}_f \;=\; \int_0^1 f(\alpha)\alpha\,d\alpha \;=\; (c-1)\int_0^\delta e^\alpha \alpha\,d\alpha \;=\; c\,\delta - 1 \;.$$

Property (ii) certainly holds for $\mu \in (\delta, 1]$. For $\mu \in [0, \delta]$ we get

$$\begin{aligned}(1+\mathrm{E}_f)\int_\mu^1 f(\alpha)\,d\alpha \;&=\; c\,\delta\,(c-1)\int_\mu^\delta e^\alpha\,d\alpha\\ &=\; c\,e^{-\gamma}\big((2-\gamma)e^\gamma - e^\mu\big)\\ &=\; c\,(2-\gamma-e^{\mu-\gamma})\\ &\leqslant\; c\,\big(2-\gamma-(1+\mu-\gamma)\big)\\ &=\; c\,(1-\mu)\;.\end{aligned}$$

This completes the proof of the lemma. □

Proof (of Theorem 12*).* The analysis of the expected completion time of job j in the random $\boldsymbol{\alpha}$-schedule follows the line of argument developed in the proof of Theorem 11. First we consider a fixed choice of α_j and bound the corresponding conditional expectation $\mathrm{E}_{\alpha_j}[C_j^{\boldsymbol{\alpha}}]$. In a second step we bound the unconditional expectation $\mathrm{E}[C_j^{\boldsymbol{\alpha}}]$ by integrating the product $f(\alpha_j)\mathrm{E}_{\alpha_j}[C_j^{\boldsymbol{\alpha}}]$ over the interval $[0, 1]$.

For a fixed job j and a fixed value α_j, the bound in (13) and Lemma 13 (i) yield

$$\begin{aligned}\mathrm{E}_{\alpha_j}[C_j^{\boldsymbol{\alpha}}] \;&\leqslant\; C_j^P(0) \;+\; (c-1)\sum_{k\in J_1}\eta_k\,p_k \;+\; \sum_{\substack{k\in J_2\\ \alpha_j>\mu_k}}(1+\mathrm{E}_f)\,p_k \;+\; (1+\alpha_j)p_j\\ &\leqslant\; c\,C_j^P(0) \;+\; (1+\mathrm{E}_f)\sum_{\substack{k\in J_2\\ \alpha_j>\mu_k}} p_k \;+\; (1+\alpha_j)p_j \;.\end{aligned}$$

The last inequality follows from (11). Using property (ii) and equation (12) yields

$$\begin{aligned}\mathrm{E}[C_j^{\boldsymbol{\alpha}}] \;&\leqslant\; c\,C_j^P(0) \;+\; (1+\mathrm{E}_f)\sum_{k\in J_2} p_k\int_{\mu_k}^1 f(\alpha_j)\,d\alpha_j \;+\; (1+\mathrm{E}_f)\,p_j\\ &\leqslant\; c\,C_j^P(0) \;+\; c\sum_{k\in J_2}(1-\mu_k)\,p_k \;+\; c\,p_j \;=\; c\,C_j^{LP} \;.\end{aligned}$$

The result follows from linearity of expectations. □

One can further say that Algorithm RANDOM-$\boldsymbol{\alpha}$ actually produces a schedule that is simultaneously expected to be near-optimal with respect to both the total weighted completion time objective and the maximum completion time objective. Bicriteria results of similar spirit and also other results in this direction have been presented in [7, 11, 47].

Corollary 6. *Under the assumptions of Theorem 12, the expected makespan of the schedule constructed by Algorithm* RANDOM-$\boldsymbol{\alpha}$ *is at most* 1.5166 *times the optimal makespan.*

Proof. The makespan of the preemptive list schedule P exactly equals the optimal makespan. Note that the expected makespan of the schedule constructed by Algorithm RANDOM-$\boldsymbol{\alpha}$ can be bounded by the sum of the idle time that already existed in the preemptive schedule plus the idle time caused by jobs $k \in J$ plus their processing times. The corollary now immediately follows from the fact that the expected idle time caused by any job k is bounded by $\mathrm{E}_f\, p_k \leqslant 0.5166\, p_k$. □

The performance of the 2-approximation algorithm with only one α given in Theorem 10 b) can also be improved through a more intricate density function; details can be found in [17].

Theorem 13. *If* α *is randomly chosen from* $[0,1]$ *according to an appropriate truncated exponential density function, then Algorithm* RANDOM-α *achieves expected performance guarantee* 1.7451. *In particular, Algorithm* BEST-α *has performance ratio* 1.7451.

7.2 A $4/3$-Approximation Algorithm for $1|\, r_j,\, pmtn \,|\sum w_j C_j$

In this subsection we prove the following theorem of Schulz and Skutella [36].

Theorem 14. *Let the* α_j*'s be chosen from a probability distribution over* $[0,1]$ *with the density function*

$$f(\alpha) \;=\; \begin{cases} \frac{1}{3}\,(1-\alpha)^{-2} & \text{if } \alpha \in [0, \frac{3}{4}], \\ 0 & \text{otherwise,} \end{cases}$$

see Figure 5. Then, the expected completion time of every job j *in the preemptive schedule constructed by the preemptive variant of Algorithm* RANDOM-$\boldsymbol{\alpha}$ *is at most* $4/3\, C_j^{LP}$ *and the expected performance guarantee of the preemptive variant of Algorithm* RANDOM-$\boldsymbol{\alpha}$ *is* $4/3$.

Notice that, in contrast to the non-preemptive case discussed in Theorem 12, we do not require the random variables α_j, $j \in J$, to be independent but allow any correlation between them. In particular, the performance ratio of $4/3$ is achieved by Algorithm BEST-α and by Algorithm RANDOM-α when α is drawn from $[0,1]$ with the density function given in Theorem 14. This result of Schulz and Skutella [36] improves upon a 1.466-approximation by Goemans, Wein, and Williamson [18]. They also analyzed Algorithm RANDOM-α with a density function similar to the one given in Theorem 14.

Again, the bound in Theorem 14 yields also a bound on the quality of the relaxations (LP):

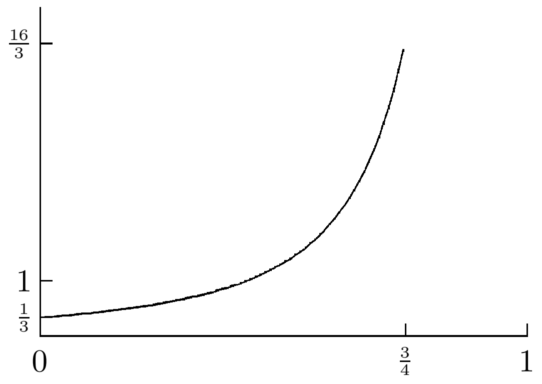

Fig. 5. The density function used for $1|\, r_j,\, pmtn\,|\sum w_j C_j$ in Theorem 14

Corollary 7. *The relaxation (LP) is a $4/3$-relaxation of $1|\, r_j,\, pmtn\,|\sum w_j C_j$.*

Following the lines of the last subsection, we state two properties of the density function f that are crucial for the analysis of the corresponding preemptive random $\boldsymbol{\alpha}$-schedule.

Lemma 14. *The function f given in Theorem 14 is a density function with the following properties:*

(i) $(1-\eta)\displaystyle\int_0^{\eta} f(\alpha)\, d\alpha \;\leqslant\; \frac{1}{3}\,\eta$ *for all* $\eta \in [0,1]$,

(ii) $\displaystyle\int_{\mu}^{1} f(\alpha)\, d\alpha \;\leqslant\; \frac{4}{3}\,(1-\mu)$ *for all* $\mu \in [0,1]$.

Similar to the situation in the last subsection, property (i) is used to bound the delay to job j caused by jobs in J_1 which corresponds to the first summation on the right-hand side of (14). The second summation reflects the delay to job j caused by jobs in J_2 and will be bounded by property (ii).

Proof (of Lemma 14*).* The function f is a density function since

$$\int_0^1 f(\alpha)\, d\alpha \;=\; \frac{1}{3}\int_0^{3/4} \frac{1}{(1-\alpha)^2}\, d\alpha \;=\; 1 \;.$$

In order to prove property (i), observe that for $\eta \in [0, 3/4]$

$$(1-\eta)\int_0^{\eta} f(\alpha)\, d\alpha \;=\; \frac{1}{3}\,(1-\eta)\int_0^{\eta} \frac{1}{(1-\alpha)^2}\, d\alpha \;=\; \frac{1}{3}\,\eta \;.$$

Since $f(\alpha) = 0$ if $\alpha > 3/4$, the bound also holds for $\eta > 3/4$. For the same reason, (ii) holds if $\mu > 3/4$. For $\mu \leqslant 3/4$ we get

$$\int_{\mu}^{1} f(\alpha)\, d\alpha \;=\; \frac{1}{3}\int_{\mu}^{3/4} \frac{1}{(1-\alpha)^2}\, d\alpha \;=\; \frac{4}{3} - \frac{1}{3}\,\frac{1}{1-\mu} \;.$$

A short computation shows that the latter expression is bounded by $\frac{4}{3}(1-\mu)$ which concludes the proof. □

Proof (of Theorem 14*).* For a fixed job j, the bound in (14) and Lemma 14 yield

$$\begin{aligned} \mathrm{E}[C_j^{\boldsymbol{\alpha}-pmtn}] &\leqslant C_j^P(0) + \frac{1}{3}\sum_{k\in J_1}\eta_k\,p_k + \frac{4}{3}\sum_{k\in J_2}(1-\mu_k)\,p_k + p_j \\ &\leqslant \frac{4}{3}C_j^P(0) + \frac{4}{3}\sum_{k\in J_2}(1-\mu_k)\,p_k + \frac{4}{3}p_j = \frac{4}{3}C_j^{LP} . \end{aligned}$$

The second inequality follows from (11), the last equation follows from (12). This concludes the proof by linearity of expectations. □

8 Approximations for Single Machine Scheduling with Precedence Constraints

In this section we present randomized approximation algorithms developed by Schulz and Skutella [35] for the scheduling problems $1|\,(r_j,)\,prec\,(,\,pmtn)\,|\sum w_jC_j$. The first constant-factor approximation algorithms for these problems have been given by Hall et al. [22]. They presented a $(4+\varepsilon)$-approximation algorithm for $1|\,prec\,|\sum w_jC_j$ and a 5.83-approximation algorithm for $1|\,r_j,\,prec\,|\sum w_jC_j$. Their algorithms are based on the time-indexed LP relaxation (LP') and scheduling in order of α-points for a fixed value of α. However, their definition of α-points is slightly different from ours. As already mentioned in Section 6, Schulz [34] and Hall et al. [21] improved upon these results. Building upon the work of Sidney [41], combinatorial 2-approximation algorithms for $1|\,prec\,|\sum w_jC_j$ were given by Chekuri and Motwani [10] and Margot, Queyranne, and Wang [25] (see also [19]). Correa and Schulz [12] look at the problem from a polyhedral perspective and uncover a relation between the work of Sidney and different linear programming relaxations. Woeginger [51] discusses the approximability of $1|\,prec\,|\sum w_jC_j$ and presents approximation preserving reductions to special cases.

A straightforward combination of the 2-approximation algorithm for the preemptive scheduling problem $1|\,r_j,\,prec,\,pmtn\,|\sum w_jC_j$ [21] with the conversion technique of Chekuri et al. given in Theorem 4 achieves a performance guarantee of $\frac{2e}{e-1}$. In particular, it does not improve upon the 3-approximation for the non-preemptive variant $1|\,r_j,\,prec\,|\sum w_jC_j$ given by Schulz [34].

For an arbitrary $\boldsymbol{\alpha}$ and a feasible fractional preemptive schedule, the order of $\boldsymbol{\alpha}$-points does in general not respect the precedence relations. Therefore, we only use one α for all jobs instead of individual α_j's. Then, the corresponding α-schedule is feasible if the fractional preemptive schedule is feasible.

The first result discussed in this section is a $(2+\varepsilon)$-approximation for the problems $1|\,prec\,|\sum w_jC_j$ and $1|\,r_j,\,prec,\,pmtn\,|\sum w_jC_j$. If we combine this algorithm with the conversion technique of Chekuri et al. in a somewhat more intricate way, we get a considerably improved approximation result for $1|\,r_j,\,prec\,|\sum w_jC_j$.

Again, we consider both the non-preemptive and the preemptive version of the algorithm:

```
Algorithm: RANDOM-$\alpha$
  1) Take a feasible solution to $(LP)$ and the corresponding feasible fractional
     preemptive schedule $P$.
  2) Draw $\alpha$ randomly from $[0, 1]$.
  3) Output the resulting (preemptive) $\alpha$-schedule.
```

The next theorem follows directly from Lemma 9.

Theorem 15. *Suppose that α is chosen from a probability distribution over $[0, 1]$ with density function $f(x) = 2x$. Then, for instances of $1|\,r_j,\, prec,\, pmtn\,|\sum w_j C_j$ and $1|\,prec\,|\sum w_j C_j$, the expected completion time of every job $j \in J$ in the schedule constructed by Algorithm* RANDOM-α *is bounded from above by twice its LP completion time C_j^{LP}.*

Proof. Lemma 9 and Lemma 10 a) yield for the preemptive and the non-preemptive case

$$\mathrm{E}[C_j^\alpha] \;=\; \int_0^1 f(\alpha)\,\frac{1}{\alpha}\,C_j^P(\alpha)\,d\alpha \;=\; 2\int_0^1 C_j^P(\alpha)\,d\alpha \;\leqslant\; 2\,C_j^{LP}\ .$$

This concludes the proof. □

Goemans (personal communication, June 1996) applied the very same technique as in Theorem 15 to improve the performance guarantee of 4 due to Hall et al. [22] for $1|\,prec\,|\sum w_j C_j$ to a performance ratio of 2.

As a result of Theorem 15, (LP) is a 2-relaxation for the precedence constrained scheduling problems $1|\,r_j,\, prec,\, pmtn\,|\sum w_j C_j$ and $1|\,prec\,|\sum w_j C_j$, see Table 2.

Combining the idea of Chekuri et al. from Theorem 4 with the technique demonstrated in the proof of Theorem 15, we can prove the following theorem.

Theorem 16. *Suppose that α is chosen from a probability distribution over $[0, 1]$ with density function $f(x) = e - e^x$, see Figure 6. Then, for instances of the problem $1|\,r_j,\, prec\,|\sum w_j C_j$ the expected completion time of every job $j \in J$ in the schedule constructed by Algorithm* RANDOM-α *is bounded from above by $e\,C_j^{LP}$.*

Proof. Just for the analysis of the randomized algorithm we emulate the random choice of α in the following way: First draw a new random variable β from $[0, 1]$ with density function $h(x) = e\,\frac{e^x-1}{e^x}$. Then, for fixed β, choose α from a probability distribution over the interval $[0, 1]$ with density function

$$g_\beta(x) \;=\; \begin{cases} \frac{e^x}{e^\beta-1} & \text{if } x \in [0, \beta], \\ 0 & \text{otherwise.} \end{cases}$$

The resulting probability distribution of the random variable α is described by the density function f since

$$f(\alpha) \;=\; e - e^\alpha \;=\; \int_0^1 h(\beta)\, g_\beta(\alpha)\, d\beta\ .$$

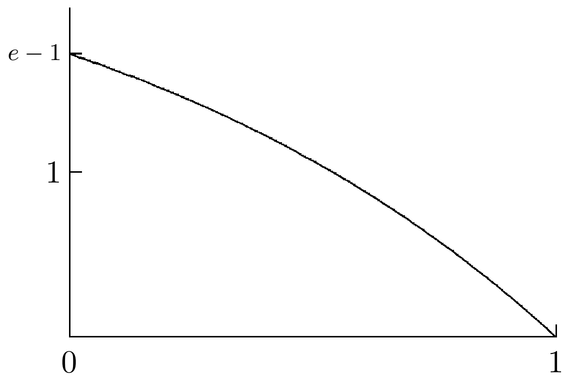

Fig. 6. The density function used for $1|\, r_j,\, prec\,|\sum w_j C_j$ in Theorem 16

For fixed $\beta \in [0,1]$ and fixed $\alpha \in [0,\beta]$ Corollary 1 and Lemma 6 yield

$$\begin{aligned} C_j^\alpha \;&\leqslant\; C_j^P(\alpha) \;+\; \sum_{\substack{k\\ \eta_k(\beta)\geqslant\alpha}} \big(1+\alpha-\eta_k(\alpha)\big)\, p_k \\ &=\; C_j^P(\alpha) \;+\; \sum_{\substack{k\\ \eta_k(\beta)\geqslant\alpha}} \big(\eta_k(\beta)-\eta_k(\alpha)\big)\, p_k \;+\; \sum_{\substack{k\\ \eta_k(\beta)\geqslant\alpha}} \big(1+\alpha-\eta_k(\beta)\big)\, p_k \\ &\leqslant\; C_j^P(\beta) \;+\; \sum_{\substack{k\\ \eta_k(\beta)\geqslant\alpha}} \big(1+\alpha-\eta_k(\beta)\big)\, p_k \;\;. \end{aligned}$$

Since

$$\int_0^\eta g_\beta(x)\,(1+\alpha-\eta)\,d\alpha \;\leqslant\; \frac{\eta}{e^\beta-1}$$

for each $\eta \in [0,1]$, the conditional expectation of j's completion time for fixed β can be bounded by

$$\begin{aligned} \mathrm{E}_\beta[C_j^\alpha] \;&\leqslant\; C_j^P(\beta) + \sum_k p_k \int_0^{\eta_k(\beta)} g_\beta(\alpha)\,\big(1+\alpha-\eta_k(\beta)\big)\,d\alpha \\ &\leqslant\; C_j^P(\beta) + \frac{1}{e^\beta-1}\sum_k \eta_k(\beta)\,p_k \;\leqslant\; \frac{e^\beta}{e^\beta-1}\,C_j^P(\beta) \;\;. \end{aligned}$$

(Notice that for $\beta = 1$ this is precisely the result of Chekuri et al., see Theorem 4.) This yields

$$\mathrm{E}[C_j^\alpha] \;=\; \int_0^1 h(\beta)\,\mathrm{E}_\beta[C_j^\alpha]\,d\beta \;\leqslant\; e\int_0^1 C_j^P(\beta)\,d\beta \;\leqslant\; e\,C_j^{LP}$$

by Lemma 10 a). $\square$

As a result of Theorem 16, (LP) is an e-relaxation for the scheduling problem $1|\, r_j,\, prec\, |\sum w_j C_j$, see Table 2. Unfortunately, the results in Theorem 15 and Theorem 16 do not directly lead to approximation algorithms for the considered scheduling problems since we cannot solve (LP) in polynomial time. However, we can overcome this drawback by introducing new variables which are not associated with exponentially many time intervals of length 1, but rather with a polynomial number of intervals of geometrically increasing size (see [37] for details). In order to get polynomial-time approximation algorithms in this way, we have to pay for with a slightly worse performance guarantee. For any constant $\varepsilon > 0$ we get randomized approximation algorithms with performance guarantee $2 + \varepsilon$ respectively $e + \varepsilon$ for the scheduling problems under consideration. Again, those algorithms can be derandomized. For details we refer to [42, Chapter 2].

9 On-Line Results

In this section we consider a certain class of on-line scheduling problems. We show that the Algorithm RANDOM-$\boldsymbol{\alpha}$, or a slightly modified version, also works in this on-line setting and the competitive ratios match the performance guarantees proved for the off-line variants. These observations are always due to the authors which also obtained the respective off-line approximation results discussed above.

In contrast to the off-line setting, an on-line algorithm does not get its entire input at one time, but receives it in partial amounts. This notion is intended to formalize the realistic scenario where we do not have access to the whole information about what will happen in the future. There are several different on-line paradigms that have been studied in the area of scheduling, see [39] for a survey. We consider the on-line setting, where jobs continually arrive over time and, for each time t, we must construct the schedule until time t without any knowledge of the jobs that will arrive afterwards. In particular, the characteristics of a job, i. e., processing time and weight are only known at its release date.

To measure the performance of a (randomized) on-line algorithm we compare the (expected) value of the schedule computed by the on-line algorithm to the value of an optimal schedule, i. e., we measure the (expected) performance of the on-line algorithms by an *oblivious adversary*, see [28, Chapter 13] for details. The worst case ratio of the two values is the *competitive ratio* of the on-line algorithm.

In order to apply RANDOM-$\boldsymbol{\alpha}$ in the on-line setting we should first mention that for each job j its random variable α_j can be drawn immediately when the job is released since there is no interdependency with any other decisions of the randomized algorithms. Moreover, PREEMPTIVE LIST SCHEDULING also works on-line if a job can be inserted at the correct position in the list with regard to the jobs that are already known as soon as it becomes available. In particular, the preemptive list schedule in order of non-decreasing ratios p_j/w_j can be constructed on-line in the first step of RANDOM-$\boldsymbol{\alpha}$ since at any point in time the ratios of all available jobs are known. Unfortunately this is not true for the $\boldsymbol{\alpha}$-points of jobs since the future development of the preemptive list schedule is not known.

However, the analysis of the non-preemptive variant of RANDOM-α still works if we schedule the jobs as early as possible in order of non-decreasing α-points, with the additional constraints that no job may start before its α-point. Notice that the non-preemptive schedule constructed by α-CONVERSION fulfills these constraints. The presented analysis of the non-preemptive variant of RANDOM-α in the proof of Theorem 12 relies on the bound given in Corollary 1 which also holds for the schedule constructed with α-CONVERSION by Lemma 7. Thus, we get an on-line variant of RANDOM-α with competitive ratio 1.6853 for $1|\, r_j \,|\sum w_j C_j$ if we replace the last step of the algorithm with the list scheduling routine described above.

This competitive ratio beats the deterministic on-line lower bound 2 for the weaker scheduling problem $1|\, r_j \,|\sum C_j$ [23]. Of course, against an oblivious adversary, a randomized algorithm can attain a substantially better competitiveness than any deterministic algorithm. The oblivious adversary cannot guess the random decisions and is therefore not able to adapt its own strategy completely to the behavior of the randomized on-line algorithm.

The deterministic 2-approximation algorithm of Phillips et al. for the scheduling problem $1|\, r_j \,|\sum C_j$ also works on-line and is therefore optimal. For the same problem Stougie and Vestjens proved the lower bound $\frac{e}{e-1}$ for randomized on-line algorithms [48, 50]. In particular, the on-line version of the algorithm of Chekuri et al. discussed in Section 4 is optimal. Recently, Anderson and Potts [3] gave a deterministic on-line algorithm for the problem $1|\, r_j \,|\sum w_j C_j$ with optimal competitive ratio 2.

The preemptive variant of RANDOM-α works without modifications in the on-line model. Notice that at any point in time and for an arbitrary pair of already available jobs we can predict whether $C_j(\alpha_j)$ will be smaller than $C_k(\alpha_k)$ or not. If one or even both values are already known we are done. Otherwise, the job with higher priority in the ratio list of the first step, say j, will win since job k cannot be processed in the list schedule P before j is finished. This yields a randomized on-line approximation algorithm with competitive ratio $\frac{4}{3}$ for the problem $1|\, r_j, pmtn \,|\sum w_j C_j$. Notice that the (deterministic) list scheduling algorithm discussed in Lemma 5 also works on-line and has competitive ratio 2. On the other hand, Epstein and van Stee [14] recently obtained lower bounds of 1.073 and 1.038 for deterministic and randomized on-line algorithms, respectively.

References

1. F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–43, New York City, NY, 1999.
2. J. M. van den Akker. *LP–Based Solution Methods for Single–Machine Scheduling Problems*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 1994.
3. E. J. Anderson and C. N. Potts. On-line scheduling of a single machine to minimize total weighted completion time. *Mathematics of Operations Research*, 29:686–697, 2004.
4. K. R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York, 1974.

5. C. C. B Cavalcante, C. C. de Souza, M. W. P. Savelsbergh, Y. Wang, and L. A. Wolsey. Scheduling projects with labor constraints. *Discrete Applied Mathematics*, 112:27–52, 2001.
6. S. Chakrabarti and S. Muthukrishnan. Resource scheduling for parallel database and scientific applications. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 329–335, Padua, Italy, 1996.
7. S. Chakrabarti, C. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 646–657. Springer, Berlin, 1996.
8. L. M. A. Chan, A. Muriel, and D. Simchi-Levi. Parallel machine scheduling, linear programming and list scheduling heuristics. *Operations Research*, 46:729–741, 1998.
9. C. S. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. R. Rau, and M. Schlansker. Profile–driven instruction level parallel scheduling with applications to super blocks. In *Proceedings of the 29th Annual International Symposium on Microarchitecture*, pages 58–67, Paris, France, 1996.
10. C. S. Chekuri and R. Motwani. Precedence constrained scheduling to minimize weighted completion time on a single machine. *Discrete Applied Mathematics*, 98:29–38, 1999.
11. C. S. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31:146–166, 2001.
12. J. R. Correa and A. S. Schulz. Single machine scheduling with precedence constraints. In D. Bienstock and G. Nemhauser, editors, *Integer Programming and Combinatorial Optimization*, volume 3064 of *Lecture Notes in Computer Science*, pages 283–297. Springer, Berlin, 2004.
13. M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26:255–270, 1990.
14. L. Epstein and R. van Stee. Lower bounds for on-line single machine scheduling. *Theoretical Computer Science*, 299:439–450, 2003.
15. M. X. Goemans. A supermodular relaxation for scheduling with release dates. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 288–300. Springer, Berlin, 1996.
16. M. X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 591–598, New Orleans, LA, 1997.
17. M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*, 15:165–192, 2002.
18. M. X. Goemans, J. Wein, and D. P. Williamson. A 1.47-approximation algorithm for a preemptive single-machine scheduling problem. *Operations Research Letters*, 26:149–154, 2000.
19. M. X. Goemans and D. P. Williamson. Two-dimensional Gantt charts and a scheduling algorithm of Lawler. *SIAM Journal on Discrete Mathematics*, 13:281–294, 2000.
20. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
21. L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.
22. L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 142–151, Atlanta, GA, 1996.

23. J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 404–414. Springer, Berlin, 1996.
24. J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In W. R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, New York, 1984.
25. F. Margot, M. Queyranne, and Y. Wang. Decomposition, network flows and a precedence constrained single machine scheduling problem. *Operations Research*, 51:981–992, 2003.
26. R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49:330–350, 2003.
27. R. H. Möhring, A. S. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM*, 46:924–942, 1999.
28. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
29. C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.
30. C. N. Potts. An algorithm for the single machine sequencing problem with precedence constraints. *Mathematical Programming Studies*, 13:78–87, 1980.
31. M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58:263–285, 1993.
32. M. W. P. Savelsbergh, R. N. Uma, and J. M. Wein. An experimental study of LP-based approximation algorithms for scheduling problems. In *Proceedings of the 9th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 453–462, San Francisco, CA, 1998.
33. A. S. Schulz. *Polytopes and scheduling*. PhD thesis, TU Berlin, Germany, 1996.
34. A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 301–315. Springer, Berlin, 1996.
35. A. S. Schulz and M. Skutella. Random-based scheduling: New approximations and LP lower bounds. In J. Rolim, editor, *Randomization and Approximation Techniques in Computer Science*, volume 1269 of *Lecture Notes in Computer Science*, pages 119–133. Springer, Berlin, 1997.
36. A. S. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5:121–133, 2002.
37. A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15:450–469, 2002.
38. P. Schuurman and G. J. Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2:203–213, 1999.
39. J. Sgall. On-line scheduling — a survey. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231. Springer, Berlin, 1998.
40. D. B. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.
41. J. B. Sidney. Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research*, 23:283–298, 1975.
42. M. Skutella. *Approximation and randomization in scheduling*. PhD thesis, Technische Universität Berlin, Germany, 1998.
43. M. Skutella and M. Uetz. Scheduling precedence-constrained jobs with stochastic processing times on parallel machines. In *Proceedings of the 12th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 589–590, Washington, DC, 2001.

44. M. Skutella and M. Uetz. Stochastic machine scheduling with precedence constraints. *SIAM Journal on Computing*, 2005. To appear.
45. W. E. Smith. Various optimizers for single-stage production. *Naval Research and Logistics Quarterly*, 3:59–66, 1956.
46. J. P. Sousa. *Time indexed formulations of non-preemptive single-machine scheduling problems*. PhD thesis, Université Catholique de Louvain, 1989.
47. C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21:115–122, 1997.
48. L. Stougie and A. P. A. Vestjens. Randomized algorithms for on-line scheduling problems: How low can't you go? *Operations Research Letters*, 30:89–96, 2002.
49. R. N. Uma and J. M. Wein. On the relationship between combinatorial and LP-based approaches to NP-hard scheduling problems. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 394–408. Springer, Berlin, 1998.
50. A. P. A. Vestjens. *On-line machine scheduling*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1997.
51. G. J. Woeginger. On the approximability of average completion time scheduling under precedence constraints. *Discrete Applied Mathematics*, 131:237–252, 2003.
52. L. A. Wolsey. Mixed integer programming formulations for production planning and scheduling problems, 1985. Invited talk at the 12th International Symposium on Mathematical Programming, MIT, Cambridge.

Approximation Algorithms for the *k*-Median Problem

Roberto Solis-Oba

Department of Computer Science,
The University of Western Ontario,
London, Ontario, Canada
solis@csd.uwo.ca

Abstract. The k-median problem is a central problem in Operations Research that has captured the attention of the Algorithms community in recent years. Despite its importance, a non-trivial approximation algorithm for the problem eluded researchers for a long time. Remarkably, a succession of papers with ever improved performance ratios have been written in the last couple of years. We review some of the approaches that have been used to design approximation algorithms for this problem, and we also present some of the known results about the hardness of approximating the optimum solution for the k-median problem.

1 Introduction

The problem of locating a set of facilities so that they can efficiently serve a group of clients has been extensively studied because of its large number of applications and interesting algorithmic aspects. Facility location problems find applications in areas as diverse as Operations Research [18,29], network design [3,19,20], data mining [9], clustering analysis [40], and web access [32,22,44], among others. These problems have been studied for over four decades [29], and many different approaches have been proposed for solving them [5,6,12,14,18,33,42].

Given sets of facilities F and clients D, the cost of servicing a client i by a facility j, denoted as $c(i,j)$, expresses the effort required to serve i from j. So, the service cost might represent the distance from a client to a facility, or the cost of shipping some commodity produced at a facility site to the client site. The most studied facility location problems are the k-median problem and the facility location problem.

In the k-median problem the goal is to select or open a set of at most k facilities (called *centers* or *medians*) that serve the clients D at minimum total cost. Given a set of centers S, a client i is always served by the center $j \in S$ for which the service cost $c(i,j)$ is minimum. The facility location problem assigns a cost $f(j)$ for opening a facility $j \in F$. The problem is to choose a set of facilities and assign clients to facilities so that the total service cost plus the total cost of the facilities is minimized. Numerous variants of these problems have been proposed in the literature, like the capacitated facility location [15], the fault tolerant k-center [21,28], bounded facility location [31], capacitated facility location with client demands [15], k-capacitated facility location [23,5], universal facility location [37], and multilevel facility location [10,49].

Let n denote the number of clients plus the number of facilities. The k-median problem is NP-hard even if the sets of facilities and clients are points on the plane and the service cost is the Euclidean distance from a client to the nearest center [26,36,43]. Moreover, the problem cannot be approximated within any factor $\alpha > 1$ of the optimum unless P=NP, even if the cost matrix is symmetric [33]. For any given value $\alpha > 1$ no polynomial time algorithm that chooses up to $o(\log n)k$ centers can approximate the value of the optimum solution for the k-median problem within a factor α of the optimum, unless P=NP [33].

Due to the difficulty of approximating the solution of the general k-median problem, many constrained versions have been considered. The most studied version of the problem is the *metric* k-median problem. In this version of the problem we interpret the cost function c as measuring the distance between a given client i and facility j. Furthermore, the cost function is extended so that it not only gives the distance between a client and a facility, but it also gives the distance between two clients or between two facilities. In the metric k-median problem, the cost function is assumed to be symmetric, i.e. $c(i,j) = c(j,i)$, and it satisfies the triangle inequality, i.e., $c(i,k) \leq c(i,j)+c(j,k)$ for any $i, j, k, \in D \cup F$. The metric facility location problem is defined analogously, and the best known algorithm for it achieves a performance ratio of 1.52 [38].

Despite the fact that the k-median problem is a central problem in Operations Research, a non-trivial approximation algorithm for it eluded researchers for a long time. The problem restricted to trees, though, has been known to be polynomially solvable for some time [25,47]. It has been fascinating to see how in the span of a few years successively better algorithms have been designed for the problem. The first approximation algorithm for the problem was designed by Lin and Vitter [33] in 1992. They showed that a $(1 + \varepsilon)$-approximation to the value of the optimum solution for the k-median problem can be computed in polynomial time if it is allowed to select up to $(1 + \frac{1}{\varepsilon})(\ln n + 1)k$ centers, for any value $\varepsilon > 0$. This result is best possible, up to constant factors, unless P=NP. For the metric k-median problem, Lin and Vitter also designed an algorithm that finds a solution of value at most $2(1+\varepsilon)$ times the optimum while selecting at most $(1 + \frac{1}{\varepsilon})k$ centers.

In 1998, Korupolu, Plaxton, and Rajaraman [30] designed a simple local search algorithm that, for any value $\varepsilon > 0$, computes a $(1+\varepsilon)$-approximation to the value of the optimum solution for the metric k-median problem that selects no more than $(3 + 5/\varepsilon)k$ centers. This algorithm can be modified so that it uses only $(3 + \varepsilon)k$ centers, but by selecting fewer centers, the value of the solution that it finds can be up to $1 + 5/\varepsilon$ times the optimum.

The first approximation algorithm for the metric k-median problem that produced a solution with at most k centers was derived in 1996 from the powerful result of Bartal [7,8], that shows how to approximate any finite metric space by a tree space. Combining this result with known algorithms for the k-median problem on trees, Bartal gave a randomized $O(\log n \log \log n)$-approximation algorithm for the metric k-median problem. Shortly after, Charikar, Chekuri, Goel, and Guha [11] showed how to de-randomize Bartal's algorithm, and they also slightly improved it obtaining an $O(\log k \log\log k)$-approximation algorithm.

The next breakthrough came a few months later, in 1998, when Arora, Raghavan, and Rao [4] presented a polynomial time approximation scheme for the metric k-median problem restricted to two-dimensional Euclidean spaces. This means that the clients and facilities are points on the plane and the cost $c(i,j)$ is the Euclidean distance between i and j. A polynomial time approximation scheme is a family of algorithms $\mathcal{A}$ such that for any value $\varepsilon > 0$ an algorithm $A_\varepsilon \in \mathcal{A}$ finds in polynomial time a solution of value at most $(1+\varepsilon)$ times the optimum.

In 1999, Charikar and Guha, and independently, Tardos and Shmoys, designed the first constant factor approximation algorithm for the metric k-median problem. They combined their results and presented a linear programming based algorithm that finds a solution of value at most $6\frac{2}{3}$ times larger than the optimum. The same year Jain and Vazirani [23] improved this result by designing a primal-dual 6-approximation algorithm. Shortly after, Charikar and Guha [12] refined Jain and Vazirani's approach to obtain a 4-approximation algorithm for the problem.

The latest improvement, to the date when this paper was written, came in 2001, when Arya, Garg, Khandekar, Meyerson, Munagala, and Pandit [5] proposed a very simple local search algorithm that finds a solution of value $(3 + \frac{2}{p} + O(\varepsilon'))$ times larger than the optimum, for any value $0 < \varepsilon' = o(n^{-2})$.. The parameter $p \geq 1$ is an integer value that guides the local search procedure, and the time complexity of the algorithm depends exponentially on it.

Another work that deserves mention is the algorithm of Mettu and Plaxton [39] for a generalization of the k-median problem known as the *online metric median problem*. In this problem the centers are selected one at a time and a facility cannot be de-selected once it has been chosen. Furthermore, the number of facilities to be selected, k, is not known in advance. The algorithm of Mettu and Plaxton finds a solution of value at most a constant factor times the value of an optimum for the k-median problem (also known as the *offline* k-median problem), for any value of k.

In this paper we describe in some detail some of the aforementioned works. In Section 2 we describe some results related to the hardness of approximating the k-median problem. Then, in Section 3, we present two linear programming based algorithms. The first one is the algorithm of Lin and Vitter [33] which approximates the solution of the general k-median problem by using up to $O(\log n)k$ centers. The second algorithm, and the rest of the algorithms that we present here, is for the metric version of the problem. This algorithm by Charikar, Guha, Tardos, and Shmoys [13] is the first algorithm which achieved constant performance ratio (this means that the algorithm approximates the optimum solution within a constant factor). In Section 4 we describe a very interesting primal dual algorithm by Jain and Vazirani [23], which exploits the fact that a Lagrangian relaxation of the k-median problem yields a special instance of the uncapacitated facility location problem. Finally, in Section 5 we present a remarkable algorithm by Arya, Garg, Khandekar, Meyerson, Munagala, and Pandit [5], which uses local search heuristics to yield the best known performance ratio for the problem.

2 Hardness of the k-Median Problem

Any instance of the k-median problem can be modeled with an undirected graph $G = (V, E)$, where each vertex corresponds to either a client or a facility, and every edge (i, j) is labelled with the corresponding cost $c(i, j)$. The number of vertices in V is n and the number of edges in E is m. Using this representation, we can give a simple proof that the k-median problem is NP-hard. Consider an instance $G = (V, E), k$ of the problem in which $V = F = D$, i.e. every vertex is both a client and a facility. Such an instance has a solution of value $n - k$ if and only if G has a dominating set of size k. Since the dominating set problem is NP-hard, the claim follows.

Interestingly, another simple reduction also shows that for any computable function $\alpha(n) > 1$, the optimum solution for the k-median problem cannot be approximated within any factor $\alpha(n)$ of the optimum in polynomial time, unless P=NP.

Theorem 1. *For any computable function $\alpha(n) > 1$, the optimum solution for the k-median problem cannot be approximated within factor $\alpha(n)$ in polynomial time, unless P=NP, even if the cost function is symmetric.*

Proof. Consider an instance $G = (V, E), k$ of the k-median problem in which $V = F = D$. Build a complete graph $G' = (V, E')$ with edge weights defined as follows:

– $c(i, i) = 0$ for all $i \in V$,
– $c(i, j) = 1$ if $(i, j) \in E$, and
– $c(i, j) = \alpha(n)(n - k) + 1$, otherwise.

Assume that there is an $\alpha(n)$-approximation algorithm A for the k-median problem. If G has a dominating set of size k, then the corresponding instance of the k-median problem has a solution of value $n - k$. Note that in this case, algorithm A would choose a set S of k centers with total service cost at most $\alpha(n)(n - k)$. Since every edge $(i, j) \in E' \setminus E$ has cost $\alpha(n)(n - k) + 1$, set S must be a dominating set of size k for G.

Interestingly, the problem is also hard to approximate if we allow the selection of more than k centers. Let S^* be an optimum solution for the k-median problem and let $c(S^*)$ be the cost of this solution.

Theorem 2. *If for any value $\rho > 1$ there is a ρ-approximation algorithm A for the k-median problem that selects a set of at most $o(\log n)k$ centers, then P=NP.*

Proof. Using the approximation preserving reduction described in the previous theorem, it is possible to show that if algorithm A exists, then A is an $o(\log n)$-approximation algorithm for the minimum dominating set problem. Since Raz and Safra [45] proved that unless P = NP, the dominating set problem does not have an $o(\log n)$-approximation algorithm, then the existence of algorithm A would imply that P = NP.

Since by Theorem 1 an optimum solution for the k-median problem cannot be approximated within any factor $\rho > 1$, we mainly focus our attention on constrained versions of the problem. The most studied version of the problem is the *metric* k-median problem, in which the cost function c is symmetric and it obeys the triangle inequality. This version of the problem is still hard to solve, even if we allow the election of more than k centers:

Theorem 3. *If A is an approximation algorithm for the metric k-median problem that finds a set S of at most $o(\log n)k$ centers that serves the clients D with total service cost $c(S) \leq c(S^*)$, then P=NP.*

Proof. Given an unweighted graph $G = (V, E)$ with minimum dominating set of size k, build a complete graph $G' = (V, E')$ with cost function c satisfying:

– $c(i, j) = 1$ if $(i, j) \in E$,
– $c(i, j) =$ length of a shortest path from i to j in G, if $(i, j) \notin E$.

Since G has a dominating set of size k, the corresponding instance of the k-median problem has a solution of value $n - k$. In this case, algorithm A finds a set S with at most $f(n)k$ centers and service cost $c(S) \leq n - k$, for some function $f(n) = o(\log n)$. We observe that the set S' of vertices not adjacent to S in G has size $|S'| \leq f(n)k$. To see this, assume that $|S'| > f(n)k$. Since the vertices in S' are at distance at least 2 from S, then the service cost of S would be

$$\begin{aligned} c(S) &\geq 2|S'| + n - |S| - |S'| \\ &\geq n + |S'| - |S| \\ &\geq n + f(n)k - f(n)k \\ &= n > n - k. \end{aligned}$$

Hence, set $S \cup S'$ is a dominating set for G of size at most $2f(n)k = o(\log n)k$ and, thus, A is an $o(\log n)$-approximation algorithm for the minimum dominating set problem, which by the aforementioned result of Raz and Safra [45], implies that P=NP.

3 Linear Programming Based Algorithms

A powerful technique that has been successfully applied to design approximation algorithms for a variety of NP-hard problems consists in formulating a problem as an integer program, then, solving the linear program obtained by relaxing the integrality constraints and, finally, rounding this fractional solution to get an integer feasible solution for the original problem. In this section we present two approximation algorithms for the k-median problem that use this technique. First, we present the algorithm of Lin and Vitter [33] which introduces an elegant rounding technique known as *filtering*. This algorithm works for the general k-median problem (with an arbitrary cost function). It finds, for any value $\varepsilon > 0$, a solution of value at most $(1+\varepsilon)$ times the optimum, but it selects up to $(1+\frac{1}{\varepsilon}) \ln n$ centers. Then, in the next section we describe an algorithm by Charikar, Guha, Tardos, and Shmoys [13], which refines the filtering technique for the metric version of the problem. This was the first algorithm for the metric k-median problem that achieved constant performance ratio using at most k centers.

3.1 Integer Program Formulation

To formulate the k-median problem as an integer program, we first need to define some variables. For each facility j we define a variable y_j which takes value 1 if j is selected as a center, and it takes value 0 otherwise. For every client-facility pair (i, j) we define a variable x_{ij} which takes value 1 if and only if client i is serviced by center j. The k-median problem can be stated as the following integer program.

$$\text{Minimize} \quad \sum_{j \in F} \sum_{i \in D} c(i,j) x_{ij} \tag{1}$$

$$\begin{aligned}
&\text{subject to} \\
&\sum_{j \in F} x_{ij} \ge 1 && \text{for all } \ i \in D \\
&\sum_{j \in F} y_j \le k && \qquad (2) \\
&x_{ij} \le y_j && \text{for all } \ i \in D, \ j \in F \\
&x_{ij}, \ y_j \in \{0,1\} && \text{for all } \ i \in D, \ j \in F
\end{aligned}$$

The linear program relaxation of the above formulation relaxes the last set of constraints allowing the variables y_j and x_{ij} to take fractional non-negative values. This linear program can be solved in polynomial time by using, for example, interior point methods [48,27]. Let $\hat{y} = (\hat{y}_1, \hat{y}_2 \ldots, \hat{y}_{|F|})$, $\hat{x} = (\hat{x}_{11}, \hat{x}_{12}, \ldots, \hat{x}_{|D||F|})$ be an optimum solution for the linear program. Before showing how to use $\hat{x}, \hat{y}$ to find an approximate solution for the k-median problem, we note that a solution for the linear program is completely characterized by the values of the $\hat{y}$ variables, since optimum fractional values for the $\hat{x}$ variables can be determined from $\hat{y}$ [2]. The idea is to (fractionally) assign each client i to its closest facilities. The algorithm for computing $\hat{x}$ from $\hat{y}$ is as follows.

Algorithm AssignClients

1. For each client i consider those facilities $i_1, i_2, \ldots, i_{|F|}$ with values $\hat{y}_{i_\ell} > 0$ in non-decreasing order of service cost: $c(i, i_1) \le c(i, i_2) \le \cdots \le c(i, i_{|F|})$.
2. Find the first facility i_p (in this ordering) for which $\sum_{\ell=1}^{p} \hat{y}_{i_\ell} \ge 1$.

3. Set the values of the variables $\hat{x}_{ij}$ in this manner:
   - $\hat{x}_{ii_\ell} \leftarrow \hat{y}_{i_\ell}$ for all $\ell = 1, 2, \ldots, p-1$, and
   - $\hat{x}_{ii_p} \leftarrow 1 - \sum_{\ell=1}^{p-1} \hat{y}_{i_\ell}$.

3.2 Filtering

The algorithm that we describe in this section approximates the value of the optimum solution for the k-median problem within a factor of $(1+\varepsilon)$, for any value $\varepsilon > 0$. The precision parameter ε affects the number of centers that the algorithm chooses, as we describe below.

The idea of filtering is to use the solution of the linear program to discard, or filter out, some of the possible assignments of clients to centers. This filtering has to be done in such a way that the problem is simplified by reducing the possible number of assignments of clients to facilities, but it has to ensure that at least one good assignment of clients to centers remains.

For each client i let $\hat{C}_i = \sum_{j \in F} c(i,j)\hat{x}_{ij}$ be the fractional cost of servicing i. We filter out some possible assignments of i to facilities by allowing client i to be served only by those facilities j such that $c(i,j) \leq (1+\varepsilon)\hat{C}_i$. Let the *neighbourhood* V_i of a client i be the set of facilities that satisfy the above condition, i.e. $V_i = \{j \mid j \in F \text{ and } c(i,j) \leq (1+\varepsilon)\hat{C}_i\}$.

We show that the neighbourhood of every client is non-empty and, therefore, there is a solution for the k-median problem in which every client is assigned to some facility in its neighbourhood.

Lemma 1. *For every client i, its neighbourhood V_i is non-empty and, furthermore,*

$$\sum_{j \in V_i} \hat{y}_j \geq \frac{\varepsilon}{1+\varepsilon}.$$

Proof. We can show that the neighbourhood V_i of a client i is not empty by using a weighted average argument. If V_i is empty, then for every facility $j \in F$, the service cost c_{ij} is larger than $(1+\varepsilon)\hat{C}_i$. Thus, $\hat{C}_i = \sum_{j \in F} c(i,j)\hat{x}_{ij} > (1+\varepsilon)\hat{C}_i \sum_{j \in F} \hat{x}_{ij} \geq (1+\varepsilon)\hat{C}_i$, by the first constraint of the integer program.

The second part of the lemma can also be proven by using a weighted average argument. Assume that $\sum_{j \in V_i} \hat{y}_i \leq \varepsilon/(1+\varepsilon)$. Then, by the third constraint of the integer program, $\sum_{j \in V_i} \hat{x}_{ij} \leq \varepsilon/(1+\varepsilon)$ and $\sum_{i \notin V_i} \hat{x}_{ij} > 1/(1+\varepsilon)$. Multiplying both sides of the last inequality by $\hat{C}_i$, we get: $\hat{C}_i < (1+\varepsilon)\hat{C}_i \sum_{j \notin V_i} \hat{x}_{ij} < \sum_{j \notin V_i} \hat{x}_{ij} c_{ij} < \hat{C}_i$.

The advantage of assigning each client to a center in its neighbourhood is that, then, the cost of servicing a client is close to the cost of servicing the client in an optimum solution for the problem.

Lemma 2. *If every client i is served by a center j in its neighbourhood V_i, then the cost of the solution is at most $(1+\varepsilon)$ times the cost $c(S^*)$ of an optimum solution for the k-median problem.*

Proof. If it is possible to find a solution x, y for the k-median problem in which clients are assigned to centers in their neighborhoods, then, for every client i there is one facility j for which $x_{ij} = 1$. Therefore, the cost of this solution is

$$\sum_{j \in F} \sum_{i \in D} c(i,j)x_{ij} \leq (1+\varepsilon) \sum_{i \in D} \hat{C}_i \leq (1+\varepsilon) \sum_{j \in F} \sum_{i \in D} c(i,j)\hat{x}_{ij} \leq (1+\varepsilon)c(S^*).$$

According to this lemma, the problem is, then, how to select for each client i a center j from its neighbourhood V_i so that the total number of centers selected is small. This latter problem is an instance of the set covering problem in which the ground set is D, the set of clients, and the family of subsets is $\{S_1, S_2, \ldots, S_{|F|}\}$, where $S_j = \{i \mid i \in D \text{ and } c(i,j) \leq (1+\varepsilon)\hat{C}_i\}$.

The greedy set cover algorithm [16,24,35], can be used to approximately solve the set covering problem, and it finds a set S of centers of size at most $\bar{s}(\ln n + 1)$, where $\bar{s}$ is the value of a fractional set cover for the above set cover problem. The value of $\bar{s}$ can be easily obtained from the solution $\hat{x}, \hat{y}$ of the linear program since by Lemma 1, for each client i, $\sum_{j \in V_i} \hat{y}_j > \varepsilon/(1+\varepsilon)$. Therefore, setting $\bar{y}_j = (1 + \frac{1}{\varepsilon})\hat{y}_j$ for every facility j yields a fractional solution for the set cover problem since, then, for each client i:

$$\sum_{i \in S_j} \bar{y}_j = \sum_{j \in V_i} \left(1 + \frac{1}{\varepsilon}\right) \hat{y}_j > 1.$$

The value of this fractional set cover $\bar{y}$ is

$$\sum_{j \in F} \bar{y}_j = \left(1 + \frac{1}{\varepsilon}\right) \sum_{j \in F} \hat{y}_j \leq \left(1 + \frac{1}{\varepsilon}\right) k.$$

The last inequality follows from the second constraint of the integer program. The algorithm is as follows.

Algorithm FILTERING

1. Solve the linear program relaxation of the k-median problem to get a fractional solution $\hat{y}, \hat{x}$.
2. For each client i, compute $\hat{C}_i = \sum_{j \in F} c(i,j)\hat{x}_{ij}$.
3. Use the greedy set cover algorithm on the instance with ground set D, and family of subsets $\{S_1, S_2, \ldots, S_{|F|}\}$, where each set S_j is formed by the clients i such that $c(i,j) \leq (1+\varepsilon)\hat{C}_i$.
4. Choose as centers those facilities selected by the greedy set cover algorithm and assign each client i to one of its closest centers.

Theorem 4. *For any value $\varepsilon > 0$, the above algorithm finds a solution of value at most $(1+\varepsilon)$ times the value of an optimum solution for the k-median problem, and this solution has at most $(1 + \frac{1}{\varepsilon})(\ln n + 1)k$ centers.*

Proof. The theorem follows from Lemma 2 and the above discussion on the greedy set cover algorithm.

3.3 An Algorithm with Constant Performance Ratio

We describe now an algorithm for the metric k-median problem with constant performance ratio. This algorithm was designed by Charikar, Guha, Tardos, and Shmoys [13], and it uses a more sophisticated version of the filtering technique than that described in the previous section. This algorithm works for the more general version of the problem where each one of the clients i has a non-negative demand d_i, and the objective is to minimize the total service cost weighted by the demands: $\min\limits_{S \subseteq F}\{\sum\limits_{i \in D} d_i \times \min\limits_{j \in S}\{c(i,j)\} \mid |S| \leq k\}$.

We describe here the version of the problem when the set of clients D and the set of facilities F are the same. Or in other words, we consider that there is a set $N = D \cup F$ of locations; each location i has some demand d_i and in each location it is possible to build a center. The distance between locations i and j is $c(i,j)$.

The linear program relaxation of the integer program formulation of this problem is as follows.

$$\text{Minimize} \quad \sum_{i,j \in N} d_i c(i,j) x_{ij} \tag{3}$$

$$\begin{aligned} &\text{subject to} \\ &\sum_{j \in N} x_{ij} \geq 1 && \text{for all } \; i \in N \\ &\sum_{j \in N} y_j \leq k && (4) \\ &x_{ij} \leq y_j && \text{for all } \; i,j \in N \\ &0 \leq x_{ij}, y_j && \text{for all } \; i,j \in N \end{aligned}$$

The problem defined by this linear program is known as the *fractional k-median problem with demands.* Given an optimum solution $\hat{x}, \hat{y}$ for this linear program, the fractional service cost of a location i is $\hat{C}_i = \sum_{j \in N} c(i,j)\hat{x}_{ij}$. Let us assume that the locations are indexed non-decreasingly by fractional service cost, so $\hat{C}_1 \leq \hat{C}_2 \leq \cdots \leq \hat{C}_{|N|}$. The algorithm is as follows.

Algorithm CONSOLIDATE

```
1. Find an optimum solution $\hat{x}, \hat{y}$ for linear program (3).
2. Re-assign demands as follows.
   Set $d'_i \leftarrow d_i$ for each location $i$.
   For each location $i$:
       If there is a location $j$ with fractional service cost $\hat{C}_j < \hat{C}_i$ and such that
       $d'_j > 0$ and $c(i,j) \leq 4\hat{C}_j$, then move the demand of $i$ to location $j$:
           $d'_j \leftarrow d'_j + d'_i$,
           $d'_i \leftarrow 0$.
```

3. Let $N'_> = \{i \in N \mid d'_i > 0\}$, be the set of locations with positive demand. We show below that $|N'_>| \leq 2k$. For each location $i \in N$, find the location $s(i) \in N'_>$ closest to it. In case of ties, choose the location with minimum index.
4. Set $x' \leftarrow \hat{x}$ and $y' \leftarrow \hat{y}$.
   For each location i with $y'_i > 0$ and $d'_i = 0$:
   - Set $y'_{s(i)} \leftarrow \min\{1, y'_i + y'_{s(i)}\}$, and set $y'_i \leftarrow 0$.
   - For each location $j \in N$, set $x'_{j\,s(i)} \leftarrow x'_{j\,s(i)} + x'_{ji}$, and set $x'_{ji} \leftarrow 0$.
5. Sort the locations $i \in N'_>$ in non-increasing order of value $d'_i\, c(i, s(i))$.
   Set $\bar{y}_i \leftarrow 1$ for the first $2k - |N'_>|$ locations $i \in N'_>$.
   Set $\bar{y}_i = \frac{1}{2}$ for the remaining $2(|N'_>| - k)$ locations $i \in N'_>$.
   Set $\bar{y}_i \leftarrow 0$ for all locations $i \in N \setminus N'_>$.
   For each location $i \in N$:
   Set $\bar{x}_{ii} \leftarrow \bar{y}_i$, and $\bar{x}_{is(i)} \leftarrow 1 - \bar{y}_i$.
6. Build a graph $H = (V_H, E_H)$ having as vertices the locations $i \in N'_>$.
   For every vertex i with $\bar{y}_i = \frac{1}{2}$ add an edge between $s(i)$ and i. Note that graph H is a forest.
   Find a dominating set I for H containing all vertices i with $\bar{y}_i = 1$, and such that $|I| \leq k$.
7. Select I as the set of centers. Let each client i be served by the center $j \in I$ with minimum service cost $c(i, j)$.

This algorithm first simplifies the problem by moving demands so that there are at most $|N'_>| \leq 2k$ locations with positive demands. These locations are far away from each other, and this allows the possibility of finding a $\frac{1}{2}$-integral solution for the problem (a solution in which every variable has value either 0, $\frac{1}{2}$, or 1). This solution can, then, be transformed into an integral solution by rounding the variables with value $\frac{1}{2}$ up to 1 or down to 0. Note that step 4 of the algorithm is not needed, we include it only to simplify the analysis. We analyze now the algorithm in more detail.

3.4 Analysis

First, we show that the re-assignment of demands does not increase the cost of the solution.

Lemma 3.

$$\sum_{i,j \in N} d'_i\, c(i,j)\hat{x}_{ij} \leq \sum_{i,j \in N} d_i\, c(i,j)\hat{x}_{ij}.$$

Proof. Since $\sum_{i,j \in N} d_i\, c(i,j)\hat{x}_{ij} = \sum_{i \in N} d_i \hat{C}_i$, and in step 2 demand is moved from location i to location j only if $\hat{C}_j \leq \hat{C}_i$, the claim follows.

Re-assigning centers as described in step 4 increases the value of the fractional solution by at most a factor of 2.

Lemma 4.

$$\sum_{i,j \in N} d'_i\, c(i,j)x'_{ij} \leq 2 \sum_{i,j \in N} d'_i\, c(i,j)\hat{x}_{ij}.$$

Proof. Note that x', y' is a feasible solution for linear program (3). Consider some location $i \in N'_>$ that is (partially) served in solution $\hat{x}, \hat{y}$ by some location $j \notin N'_>$. The (fractional) center at j is moved to location $s(j)$ and, hence, the service cost of i increases to $c(i, s(j))\, \hat{x}_{ij} \leq (c(i,j) + c(j, s(j)))\hat{x}_{ij} \leq 2\, c(i,j)\, \hat{x}_{ij}$, since by definition of $s(j)$, $c(j, s(j)) \leq c(j,i) = c(i,j)$. Therefore, the total service cost is increased by at most a factor of 2.

We show that the solution x', y' built in step 4 is such that at least one half of a center is assigned to each location $i \in N'_>$, i.e., $y'_i \geq \frac{1}{2}$. By constraint (4) of the linear program (3), this means that set $N'_>$ has at most $2k$ locations.

Lemma 5. *For each location $i \in N'_>$, $y'_i \geq \frac{1}{2}$.*

Proof. For any two locations $i, j \in N'_>$, $c(i,j) > 4\max\{\hat{C}_i, \hat{C}_j\}$ because otherwise the demands for i and j would have been merged in step 2 of the algorithm. Hence, after re-assigning demands, for each location $i \in N'_>$ all (fractional) centers within distance $2\,\hat{C}_i$ of i are moved to i in step 4. Note that for any $i \in N'_>$,

$$\sum_{j:c(i,j)>2\hat{C}_i} \hat{x}_{ij} < \frac{1}{2}$$

because, otherwise

$$\sum_{j:c(i,j)>2\hat{C}_i} c(i,j)\hat{x}_{ij} > 2\hat{C}_i \sum_{j:c(i,j)>2\hat{C}_i} \hat{x}_{ij} \geq \hat{C}_1$$

which is a contradiction. Since $\sum_{j \in N} \hat{x}_{ij} \geq 1$ and $\hat{x}_{ij} \leq \hat{y}_j$ for each $i \in N$, then

$$\sum_{j:c(i,j)\leq 2\hat{C}_i} \hat{y}_j \geq \sum_{j:c(i,j)\leq 2\hat{C}_i} \hat{x}_{ij} \geq \frac{1}{2}.$$

Therefore, in step 4 of the algorithm the value assigned to each variable y'_i, $i \in N'_>$, is at least $\frac{1}{2}$.

It is easy to verify that the $\frac{1}{2}$-integral solution $\bar{x}, \bar{y}$ built in step 5 is feasible for the linear program (3) with demands d'. As for its cost, we show that it is not larger than the cost of solution x', y'.

Lemma 6.
$$\sum_{i,j \in N} d'_i c(i,j)\bar{x}_{ij} \leq \sum_{i,j \in N} d'_i c(i,j) x'_{ij}.$$

Proof.

$$\begin{aligned}
\sum_{i,j \in N} d'_i c(i,j)\bar{x}_{ij} &= \sum_{i \in N'_>} \sum_{j \in N'_>} d'_i c(i,j)\bar{x}_{ij} \\
&= \sum_{i \in N'_>} d'_i c(i, s(i))(1 - \bar{y}_i) \\
&\leq \sum_{i \in N'_>} d'_i c(i, s(i))(1 - y'_i) \qquad (5)
\end{aligned}$$

To show that the last inequality is true, let us define N'_1 as the set of centers $i \in N'_>$ for which $\bar{y}_i = 1$. Also, let $d'_\ell c(\ell, s(\ell)) = \min_{i \in N'_1} \{d'_i c(i, s(i))\}$. Because of the way in which set N'_1 is chosen in step 5 of the algorithm, we know that $d'_\ell c(\ell, s(\ell)) \geq d'_i c(i, s(i))$ for all $i \notin N'_1$. Finally, recall that $y'_i \leq 1$ for all $i \in N$. Hence,

$$\begin{aligned}
&\sum_{i \in N'_>} d'_i c(i, s(i))(1 - \bar{y}_i - (1 - y'_i)) \\
&\qquad = \sum_{i \in N'_>} d'_i c(i, s(i))(y'_i - \bar{y}_i) \\
&\qquad = \sum_{i \in N'_1} d'_i c(i, s(i))(y'_i - 1) + \sum_{i \notin N'_1} d'_i c(i, s(i))(y'_i - \frac{1}{2}) \\
&\qquad \leq \sum_{i \in N'_1} d'_\ell c(\ell, s(\ell))(y'_i - 1) + \sum_{i \notin N'_1} d'_\ell c(\ell, s(\ell))(y'_i - \frac{1}{2}) \\
&\qquad = \sum_{i \in N'_1} d'_\ell c(\ell, s(\ell))(y'_i - \bar{y}_i) + \sum_{i \notin N'_1} d'_\ell c(\ell, s(\ell))(y'_i - \bar{y}_i) \\
&\qquad = d'_\ell c(\ell, s(\ell)) \sum_{i \in N'_>} (y'_i - \bar{y}_i) \\
&\qquad = 0
\end{aligned}$$

The last equation follows from the fact that $\sum_{i \in N'_>} y'_i = \sum_{i \in N'_>} \bar{y}_i = k$. To complete the proof, we note that

$$\sum_{\substack{j \in N' \\ j \neq i}} x'_{ij} \geq 1 - x'_{ii} \geq 1 - y'_i.$$

Therefore,

$$\sum_{i \in N'_>} d'_i c(i, s(i))(1 - y'_i) \leq \sum_{i \in N'_>} \sum_{j \in N'_>} d'_i c(i, s(i)) x'_{ij} \leq \sum_{i \in N'_>} \sum_{j \in N'_>} d'_i c(i, j) x'_{ij}.$$

Combining this with inequality (5), the claim follows.

Now, we are ready to show that the algorithm has a constant performance ratio.

Theorem 5. *Algorithm* CONSOLIDATE *finds a solution of cost at most* 8 *times the cost of an optimum solution for the metric k-median problem.*

Proof. It is not hard to see that the graph H built in step 6 is a forest and, hence, we can efficiently find the required dominating set I as follows. First, add to I all those vertices i with $\bar{y}_i = 1$ and, then, remove such vertices from H. Any isolated vertex in the resulting graph H is dominated by at least one vertex in I and, thus, these vertices are also deleted from H. After these deletions, H is a forest in which every tree contains at least 2 vertices, and the total number of vertices in H is at most $|N'_>| - |I| = 2(|N'_>| - k)$. A minimum dominating set I' of H can be easily found and it includes at most half of the vertices in each tree of H. Thus, $|I'| \leq |N'_>| - k$. We set $I \leftarrow I \cup I'$ to get a dominating set for the original graph H of size $|I| \leq 2k - |N'_>| + |N'_>| - k = k$ as desired.

Consider this solution I. Each vertex $i \in I$ with $\bar{y}_i = 1$ does not contribute to the cost of the solution since $\bar{x}_{ii} = \bar{y}_i = 1$ and $c(i,i) = 0$. For those vertices i with $\bar{y}_i = \frac{1}{2}$, either $i \in I$ or $s(i) \in I$. Hence, the contribution of i to the cost of the solution is at most $d'_i\, c(s(i),i) \leq 2d'_i\, c(s(i),i)\bar{x}_{i\,s(i)}$. Therefore, the cost $c(I)$ of solution I is at most 2 times the cost of the half integral solution $\bar{x}, \bar{y}$. By Lemmas 4 and 6, $c(I)$ is at most 4 times the cost of the fractional solution $\hat{x}, \hat{y}$ with demands d'.

To determine the effect of the re-allocation of demands performed in step 2 on the cost of the solution, consider a location $j \in N$ which has its demand moved to another location $j' \in N'$. For this location $c(j',j) \leq 4\hat{C}_j$. Let j' be served by center i in solution I. If we move back the demand to j and let j be served by center i, this would increase the cost of the solution by at most $d_j\, c(j',j) \leq 4 d_j \hat{C}_j$. Therefore, moving all demands back to the original locations increases the cost of the solution by at most 4 times the cost of solution $\hat{x}, \hat{y}$.

By the above arguments, the cost of solution I is at most 8 times the cost of an optimum solution for the k-median problem.

By using a more complex rounding algorithm than the one described in step 6 of the algorithm, Charikar et al. [13] are able to show that the performance ratio of the algorithm can be improved to $6\frac{2}{3}$.

4 A Primal-Dual Algorithm

In this section we describe another linear programming based approximation algorithm for the k-median problem. The approach that we present now differs from the ones presented in the previous section in that it does not need to solve a linear program, but rather it finds primal and dual solutions for the problem using combinatorial methods. This results in a faster algorithm, and, as we show, the performance ratio is better than the performance ratio of the algorithm of Charikar et al. [13].

The primal-dual algorithm that we present here is due to Jain and Vazirani [23]. This algorithm is based on an interesting relationship between the k-median problem and the uncapacitated facility location problem. In order to understand the algorithm we need first to describe a primal-dual approximation algorithm for the uncapacitated facility location problem.

4.1 The Uncapacitated Facility Location Problem

The uncapacitated facility location problem differs from the k-median problem in that facilities have assigned building costs $f(j)$ and there is no bound on the number of facilities that might be selected as centers. The goal is to minimize the total cost for servicing the clients plus the cost of the facilities selected. An integer program defining the uncapacitated facility location problem is very similar to integer program (1), with the differences stated above. A linear program relaxation of this integer program is given below.

$$\text{Minimize} \sum_{j\in F}\sum_{i\in D} c(i,j)x_{ij} + \sum_{j\in F} f(j)y_j \tag{6}$$

$$\begin{aligned} &\text{subject to} \\ &\sum_{j\in F} x_{ij} \geq 1 && \text{for all } \ i \in D \\ &x_{ij} \leq y_j && \text{for all } \ i \in D, \ j \in F \\ &0 \leq x_{ij}, \ y_j && \text{for all } \ i \in D, \ j \in F \end{aligned}$$

The dual linear program corresponding to this linear program is the following (for a comprehensive study of linear programming concept the reader is referred to [17,41,46]).

$$\text{Maximize} \sum_{i\in D} \alpha_i \tag{7}$$

$$\begin{aligned} &\text{subject to} \\ &\alpha_i - \beta_{ij} \leq c(i,j) && \text{for all } \ i \in D, \ j \in F \\ &\sum_{i\in D} \beta_{ij} \leq f(j) && \text{for all } \ j \in F \\ &\alpha_i, \ \beta_{ij} \geq 0 && \text{for all } \ i \in D, \ j \in F \end{aligned} \tag{8}$$

There is a nice interpretation for the dual variables α, β. Let us think, as it happens in the real world, that the clients must pay for the service cost and for the cost of building the selected facilities. If client i is served by facility j, then the amount α_i paid by the client must be at least equal to $c(i,j)$. If $\alpha_i > c(i,j)$, the rest of the money paid by the client, i.e., $\beta_{ij} = \alpha_i - c(i,j)$, goes towards paying for the cost of building facility j. By the complementary slackness conditions, the second constraint of the dual linear program is *tight* (which means that $\sum_{i\in D}\beta_{ij} = f(j)$) if facility j is selected. By the above argument, the total amount $\sum_{i\in D}\beta_{ij}$ contributed by the clients served by j, has to be exactly equal to the building cost of facility j.

To solve the dual linear program, we must determine the price that each client must pay. From the dual solution it is easy to determine which facilities are selected. Each client is assigned to that facility with smallest service cost.

An instance of the uncapacitated facility location problem can be modeled with a graph $G = (D \cup F, E)$ having as vertices the clients and facilities, and edges connecting every facility to each client. The weight of an edge (i,j) is the service cost $c(i,j)$. In the above primal-dual context, an edge (i,j) is said to be *tight* if $\alpha_i \geq c(i,j)$ (or in other words, if $\alpha_i = c(i,j) + \beta_{ij}$ and $\beta_{ij} \geq 0$), and a facility j is said to be *paid for* if $\sum_{i\in D}\beta_{ij} = f(j)$. A client i is *marked* if there is a facility j which has been paid for, and edge (i,j) is tight. The algorithm for approximately solving the uncapacitated facility location problem is as follows.

Algorithm PrimalDual

```
0. Initially all clients are un-marked. All values $\alpha_i$ and $\beta_{ij}$ are initialized to 0.
1. Repeat steps 2 and 3 as long as there are un-marked clients.
2. Simultaneously and uniformly (at the same rate) raise the values of, both,
   the dual variables $\alpha_i$ for all un-marked clients $i$ and the variables $\beta_{ij}$ for all
   tight edges $(i,j)$, until either:
     – $\alpha_i = c(i,j)$ for some edge $(i,j)$, or
     – $\sum_{i \in D} \beta_{ij} = f(j)$ for some facility $j$.
   In the first case we label edge $(i,j)$ as tight. In the second case we label
   facility $j$ as paid for.
3. Every un-marked client $i$ with a tight edge $(i,j)$ connecting it to a paid for
   facility $j$ is marked.
4. Build the graph $T = (D \cup F, E')$ containing those edges $(i,j)$ for which
   $\beta_{ij} > 0$.
5. Build the square $T^2$ of graph $T$ by adding an edge between vertices $u$ and $v$
   if there is a path of length at most 2 between $u$ and $v$ in $T$.
6. Build the subgraph $H$ of $T^2$ induced by those facilities that are paid for.
7. Find a maximal independent set $I$ of $H$.
8. Select $I$ as the set of centers and let each client $i$ be served by a center
   $j \in I$ for which the service cost $c(i,j)$ is minimum. If for a client $i$ there are
   2 or more centers with minimum service cost, choose any one of them for
   servicing $i$.
```

In the solution I produced by this algorithm, we say that a client i is *directly connected* to the facility j that serves it if $\beta_{ij} > 0$. Otherwise client i is said to be *indirectly connected.*

4.2 Performance Ratio

We interpret the value of variable α_i as the price that client i has to pay for, both, building a facility and for being serviced by that facility. Let client i be serviced by facility j. Let α_i^f be the price that client i pays for building facility j and let α_i^s be the service cost paid by the client. Clearly, $\alpha_i = \alpha_i^s + \alpha_i^f$. If client i is directly connected to j, then $\alpha_i = c(i,j) + \beta_{ij}$, and so we set $\alpha_i^s \leftarrow c(i,j)$, and $\alpha_i^f \leftarrow \beta_{ij}$. If i is indirectly connected, then we set $\alpha_i^s \leftarrow \alpha_i$ and $\alpha_i^f \leftarrow 0$.

Lemma 7.
$$\sum_{j \in I} f(j) = \sum_{i \in D} \alpha_i^f .$$

Proof. Every facility $j \in I$ is paid for, i.e. $f(j) = \sum_{i \in D} \beta_{ij} = \sum_{i \in D_j} \alpha_i^f$, where D_j is the set of clients directly connected to j. Since sets D_j are disjoint, the claim follows.

Lemma 8. *For every indirectly connected client i, $c(i,j) \leq 3\alpha_i^s$, where $j \in I$ is the facility that serves i.*

Proof. Since the loop in steps 1-3 terminates when all clients are marked, then, i is marked in step 3 because these is a facility j' for which $\alpha_i \geq c(i,j')$. However, $j' \notin I$ since i is indirectly connected. As i is indirectly connected, facility j' was not selected to be in the final solution I computed in step 7. Note that since I is a maximal independent set of H, there has to be at least one facility $k \in I$ such that there is an edge from j' to k in the graph H built in step 6 (see Figure 1). Because clients are assigned to their nearest centers in step 8, then $c(i,k) \geq c(i,j)$.

Observe that k and j' are facilities and so there is no edge (k,j') in the graph $G = (D \cup F, E)$. Since edge (k,j') belongs to H, there must be a client h with edges (h,k) and (h,j') such that $\beta_{hk} > 0$ and $\beta_{hj'} > 0$. This implies that $\alpha_h > c(h,k)$ and $\alpha_h > c(h,j')$.

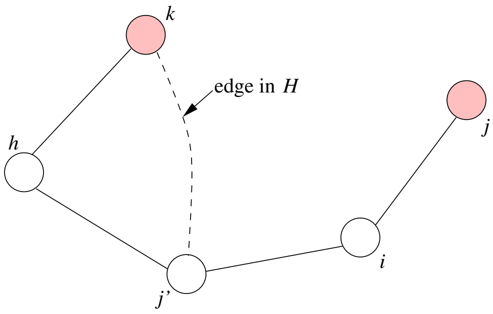

Fig. 1. Service cost of client i. Centers k and j belong to I, but $j' \notin I$.

The algorithm does not rise the value of α_h after facility j' is paid for (and in fact it might stop raising the value of α_h when k or other neighbouring facility of h is paid for). Since, as we assumed above, i is marked when α_i takes value $c(i,i')$, then α_i is raised at least until the time when j' is paid for. Therefore, $\alpha_i > \alpha_h$. By the triangle inequality (see Figure 1), $c(i,j) \leq c(i,k) \leq c(i,j') + c(h,j') + c(h,k) \leq \alpha_i + 2\alpha_h \leq 3\alpha_i$.

Using these two lemmas, we can compute the performance ratio of algorithm PrimalDual.

Theorem 6. *The performance ratio of algorithm* PrimalDual *is* 3.

Proof. Let x, y and α, β be the primal and dual solutions produced by the algorithm. Since for a client i directly connected to a facility j, $\alpha_i^s = c(i,j)$, and by Lemma 8, for each indirectly connected client i, $c(i,j) \leq 3\alpha_i^s$, then

$$\sum_{j \in F} \sum_{i \in D} c(i,j) x_{ij} \leq 3 \sum_{i \in D} \alpha_i^s.$$

From this inequality and Lemma 7, we get

$$\sum_{j\in F}\sum_{i\in D} c(i,j)x_{ij} + 3\sum_{j\in F} f(j)y_j \le 3\sum_{i\in D}(\alpha_i^s + \alpha_i^f) = 3\sum_{i\in D}\alpha_i. \tag{9}$$

Since the value of an optimum solution for the k-median problem is at least $\sum_{i\in D}\alpha_i$, the claim follows.

4.3 Running Time

Steps 1-3 are, perhaps, the most difficult steps of the algorithm to implement and analyze, so we will give some details here as to how these steps can be efficiently implemented. Note that since the values α_i are raised uniformly, the order in which the edges will become tight is consistent with a non-decreasing ordering of the edges by cost. Hence, if we store the edges in a list L in non-decreasing order of cost, we will know the order in which the events $\alpha_i = c(i,j)$ of step 2 will take place. To keep track of the second class of events that take place in step 2, namely $\sum_{i\in D}\beta_{ij} = f(j)$ for some facility j, we need to maintain for each facility j the following 3 variables:

- a variable b_j giving the number of tight edges currently contributing to facility j,
- a variable t_j giving the time when the value of b_j last changed, and
- a variable p_j giving the total contribution made by clients to facility j up to time t_j.

To determine the event that will take place during the following iteration of steps 2-3 we first compute

$$\tau = \min\left\{c(i,j), \min_{j\in F}\left\{\frac{f(j)-p_j}{b_j}\right\}\right\},$$

where (i,j) is the first edge in L, if any. If L is empty, we simply ignore the first term $c(i,j)$. We can efficiently compute τ by using a heap h.

a. If $c(i,j)=\tau$, then edge (i,j) becomes tight in this iteration, so we discard (i,j) from L.
   - If j is not yet paid for, we update $p_j \leftarrow p_j + (\tau - t_j)b_j$ and, then, increase the value of b_j by 1 since in the next iterations the value of β_{ij} will be increased. Finally, we set $t_j \leftarrow \tau$. These steps can be performed in constant time and updating the value $\frac{f(j)-p_j}{b_j}$ associated with j in the heap needs $O(\log n_f)$ time, where n_f is the number of facilities.
   - If j is already paid for, then we mark i. Since from this point on the value α_i and the values $\beta_{ij'}$ for all tight edges (i,j') will not increase any more, then, we need to update the values of the variables for such facilities j'. For each facility j' such that (i,j') is tight we set $p_{j'} \leftarrow p_{j'} + b_{j'}(\tau - t_{j'})$ and, then, we decrease $b_{j'}$ by 1. Finally, we set $t_{j'} \leftarrow \tau$. The total amount of time needed to perform these steps is $O(\text{degree}(i)\log n_f)$, where degree$(i)$ is the degree of client i in the graph $G=(D\cup F,E)$.

b. On the other hand, if $\frac{f(j)-p_j}{b_j} = \tau$, then facility j will next get paid for, so we remove this value $\frac{f(j)-p_j}{b_j}$ from the heap. Furthermore, all clients i with tight edges to j will be marked, and so their α_i and β_{ij} values will stop increasing. For each one of these clients with tight edges (i, j') we need to update the values of $p_{j'}$, $t_{j'}$, and $b_{j'}$ as described above. Let S_j be the set of clients marked in this step. The total time needed to perform the above steps is $O(\sum_{i \in S_j} \text{degree}(i) \log n_f)$.

Since every client is marked only once, then the total time needed to complete steps 1-3 is $O(\sum_{i \in D} \text{degree}(i) \log n_f) = O(m \log n_f)$, where m is the number of edges in the graph $G = (D \cup F, E)$.

4.4 Algorithm for the k-Median Problem

By studying the integer program formulations for the k-median and uncapacitated facility location problems, we note that the Lagrangian relaxation of constraint (2) in the integer program (1) for the k-median problem gives an instance of the uncapacitated facility location problem in which all the facilities have the same cost z, where z is the Lagrange multiplier:

$$\begin{aligned}
&\text{Minimize} \quad \sum_{j \in F} \sum_{i \in C} c(i,j) x_{ij} + \sum_{j \in F} z y_j && (10)\\
&\text{subject to} \\
&\quad \sum_{j \in F} x_{ij} \geq 1 && \text{for all } \; i \in D \\
&\quad x_{ij} \leq y_j && \text{for all } \; i \in D, \; j \in F \\
&\quad x_{ij}, \; y_j \in \{0,1\} && \text{for all } \; i \in D, \; j \in F
\end{aligned}$$

The value of the Lagrange multiplier z controls the number of facilities selected by the algorithm PrimalDual. As the value of z increases, the number of facilities selected decreases. If it happens that for some value of z the algorithm chooses exactly k facilities, then this would be a good solution for the k-median problem:

Lemma 9. *Suppose that for some value z algorithm* PrimalDual *selects exactly k facilities. The cost of this solution is at most* 3 *times the cost of an optimum solution for the k-median problem.*

Proof. Let x, y and α, β be the primal and dual solutions constructed by the algorithm. Then, by equation (9),

$$\sum_{j \in F} \sum_{i \in D} c(i,j) x_{ij} \leq 3 \left(\sum_{i \in D} \alpha_i - kz \right) \qquad (11)$$

Observe that x, y is a feasible solution for the k-median problem and α, β, z is a feasible solution for the dual linear program for the k-median problem:

$$\begin{aligned} &\text{Maximize} \quad \sum_{i \in D} \alpha_i - zk && && (12)\\ &\text{subject to} && && \\ &\quad \alpha_i - \beta_{ij} \le c(i,j) && \text{for all } \ i \in D,\ j \in F && \\ &\quad \sum_{i \in D} \beta_{ij} \le z && \text{for all } \ j \in F && \\ &\quad \alpha_i,\ \beta_{ij} \ge 0 && \text{for all } \ i \in D,\ j \in F && \end{aligned}$$

Therefore, by the weak duality theorem, x, y is a solution for the k-median problem of cost $\sum_{j \in F} \sum_{i \in D} c(i,j) x_{ij}$ at most 3 times the cost of an optimum solution for the problem.

However, there might not be a value z for which algorithm PrimalDual selects exactly k centers, or it might be the case that finding such a value might take too long[1]. What we can do in that case, is to find two "close"values z_1 and z_2 for the facility costs, such that for the first value the algorithm will choose $k_1 < k$ and for the second one it will select $k_2 > k$ centers, respectively. We combine these solutions to get a fractional solution with exactly k centers. The details are as follows.

Algorithm ConvexCombination

```
1. Compute $c_{\min} = \min\{c(i,j) \mid i \in D, j \in F\}$ and $c_{\max} = \max\{c(i,j) \mid i \in D, j \in F\}$.
   Set $n_f \leftarrow |F|$.
2. Use algorithm PrimalDual and binary search over the interval $[0, nc_{\max}]$
   to find two values, $z_1$ and $z_2$, such that $z_1 - z_2 \le c_{\min}/(4n_f^2)$ for which
   algorithm PrimalDual finds solutions
     – $x^s, y^s, \alpha^s, \beta^s$ with $k_1 < k$ centers, and
     – $x^\ell, y^\ell, \alpha^\ell, \beta^\ell$ with $k_2 > k$ centers, respectively.
   Let $A$ be the set of centers chosen in solution $x^s, y^s$, and let $B$ be the set of
   centers chosen in solution $x^\ell, y^\ell$.
3. Let $a = \frac{k_2 - k}{k_2 - k_1}$ and $b = \frac{k - k_1}{k_2 - k_1}$. Combine the two above solutions to get a
   new solution $(\hat{x}, \hat{y}) = a(x^s, y^s) + b(x^\ell, y^\ell)$ that (fractionally) opens exactly $k$ centers.
4. $B' \leftarrow \emptyset$
   For each facility $j \in A$ do
       Remove from $B$ the facility $j'$ with minimum $c(j, j')$ value, and include
   it into set $B'$.
```

[1] Recently Archer et al. [1] designed a variant of Jain and Vazirani's algorithm which guarantees the existence of a value k for which exactly k centers are selected. Unfortunately, this algorithm requires the solution of the maximum independent set problem, and so it is not guaranteed to run in polynomial time.

5. Choose a set I of k centers from $A \cup B$ as follows:
   – Select all centers in A with probability a and select all centers in B' with probability b (note that $b = 1 - a$).
   – Randomly select $k - k_1$ centers from B.
6. Select I as the set of centers and let each client i be served by the center $j \in I$ with minimum service cost $c(i, j)$.

Lemma 10. *The cost of the fractional solution $\hat{x}, \hat{y}$ computed in step 3 is at most $(3 + \frac{1}{n_f})$ times the cost of an optimum fractional solution for the k-median problem.*

Proof. Let $\alpha = a\alpha^s + b\alpha^\ell$, and $\beta = a\beta^s + b\beta^\ell$. Note that, α, β, z_1 is a feasible solution for the dual linear program (12) and, thus, its value is a lower bound for the value of the optimum fractional solution for the k-median problem. Let us compute the value of this dual solution. By equation (9),

$$3\left(\sum_{i\in D}\alpha_i^\ell - z_2k_2\right) \geq \sum_{j\in F}\sum_{i\in D} c(i,j)x_{ij}^\ell \geq c_{\min} \geq (3n_f + 3)(z_1 - z_2)k_2.$$

This last inequality follows since $k_2 \leq n_f$, and $z_1 - z_2 \leq c_{\min}/(4n_f^2)$. Furthermore, we need to assume that $n_f \geq 3$. This is a reasonable assumption since if the number of facilities is less than 3, then the k-median problem can be easily solved by trying the at most $O(n^2)$ subsets of k facilities as possible solutions for the problem. Hence,

$$3\left(\sum_{i\in D}\alpha_i^\ell - z_1k_2\right) = 3\left(\sum_{i\in D}\alpha_i^\ell - z_2k_2\right) - 3(z_1 - z_2)k_2 \geq 3n_f(z_1 - z_2)k_2.$$

So, $(z_1 - z_2)k_2 \leq \frac{1}{n_f}\left(\sum_{i\in D}\alpha_i^\ell - z_1k_2\right)$ and, therefore,

$$3\left(\sum_{i\in D}\alpha_i^\ell - z_2k_2\right) = 3\left(\sum_{i\in D}\alpha_i^\ell - z_1k_2\right) + (z_1 - z_2)k_2 \leq \left(3 + \frac{1}{n_f}\right)\left(\sum_{i\in D}\alpha_i^\ell - z_1k_2\right).$$

Combining the above inequalities, we get

$$\sum_{j\in F}\sum_{i\in D} c(i,j)x_{ij}^\ell \leq \left(3 + \frac{1}{n_f}\right)\left(\sum_{i\in D}\alpha_i^\ell - z_1k_2\right).$$

Also, by inequality (9),

$$\sum_{j\in F}\sum_{i\in D} c(i,j)x_{ij}^s \leq 3\left(\sum_{i\in D}\alpha_i^s - z_1k_1\right).$$

From the last two inequalities, we finally get:

$$\begin{aligned}\sum_{j\in F}\sum_{i\in D} c(i,j)\hat{x}_{ij} &\le \left(3+\frac{1}{n_f}\right)\left(a\sum_{i\in D}\alpha_i^s + b\sum_{i\in D}\alpha_i^\ell - z_1 k\right)\\ &= \left(3+\frac{1}{n_f}\right)\left(\sum_{i\in D}\alpha_i - z_1 k\right).\end{aligned}$$

Let x^I, y^I be the solution for integer program (1) corresponding to the solution I computed by algorithm ConvexCombination. The expected cost of this solution is bounded in the following Lemma.

Lemma 11.

$$E[\sum_{j\in F}\sum_{i\in D} c(i,j)x^I_{ij}] \le (1+\max\{a,b\})\sum_{j\in F}\sum_{i\in D} c(i,j)\hat{x}_{ij},$$

where $\hat{x}$ is the fractional solution computed in step 3.

Proof. Consider a client i and the sets of facilities A, B, and B' constructed by the algorithm in steps 2 and 4. Let $j_1 \in A$ and $j_2 \in B$ be the centers that serve i in solutions x^s, y^s and x^ℓ, y^ℓ, respectively. We consider two cases.

1. If $j_2 \in B'$, then either $j_1 \in I$ or $j_2 \in I$. Hence,

$$E[\sum_{j\in F} c(i,j)x^I_{ij}] = a\,c(i,j_1) + b\,c(i,j_2) = \sum_{j\in F} c(i,j)\hat{x}_{ij}.$$

2. If $j_2 \notin B'$, then
   - the probability that $j_2 \in I$ is b,
   - the probability that $j_2 \notin I$ and $j_1 \in I$ is $a(1-b) = a^2$, and
   - the probability that $j_1 \notin I$ and $j_2 \notin I$ is $(1-a)(1-b) = ab$.

   Let $j_3 \in B'$ be the facility that is paired to j_1 in step 4 of the algorithm. Then,

$$E[\sum_{j\in F} c(i,j)x^I_{ij}] \le b\,c(i,j_2) + a^2 c(i,j_1) + ab\,c(i,j_3).$$

   Since j_3 is the center in B closest to j_1, then $c(j_1,j_3) \le c(j_1,j_2)$, and so $c(i,j_3) \le c(i,j_1) + c(j_1,j_3) \le c(i,j_1) + c(j_1,j_2) \le 2c(i,j_1) + c(i,j_2)$. Using this last inequality we get:

$$\begin{aligned}E[\sum_{j\in F} c(i,j)x^I_{ij}] &\le (a^2+2ab)c(i,j_1) + (b+ab)c(i,j_2)\\ &= a(a+2b)c(i,j_1) + b(1+a)c(i,j_2)\\ &= a(1+b)c(i,j_1) + b(1+a)c(i,j_2)\\ &\le (1+\max\{a,b\})(c(i,j_1)a + c(i,j_2)b)\\ &= (1+\max\{a,b\})\sum_{j\in F} c(i,j)\hat{x}_{ij}\end{aligned}$$

We are now ready to determine the performance ratio of the algorithm.

Theorem 7. *There is a deterministic 6-approximation algorithm for the metric k-median problem.*

Proof. Since algorithm ConvexCombination can be easily de-randomized using the method of conditional expectations, we only need to show that the expected cost of the solution x^I, y^I is at most 6 times the cost of an optimum solution for the k-median problem.

By the previous two lemmas, the cost of the solution x^I, y^I is at most $(3 + \frac{1}{n_f})(1 + \max\{a, b\})$ times larger than the optimum. By using basic algebra we can show that, $a = \frac{k_2 - k}{k_2 - k_1} \leq \frac{n_f - k}{n_f - (k-1)} \leq \frac{n_f - 1}{n_f}$, and $b = \frac{k - k_1}{k_2 - k_1} \leq \frac{k-1}{k}$. Then $1 + \max\{a, b\} \leq 1 + \frac{n_f - 1}{n_f} \leq 2 - \frac{1}{n_f}$. Therefore, the performance ratio of the algorithm is $\left(3 + \frac{1}{n_f}\right)\left(2 - \frac{1}{n_f}\right) \leq 6$.

Charikar and Guha [12] proposed a slight variation of the above algorithm, and using a more complex analysis they were able to show that their algorithm has a performance ratio of 4.

5 A Local Search Algorithm

Local search heuristics have not been widely used to design approximation algorithms, mainly because of the difficulty of proving that a locally optimal solution is within a certain factor of the globally optimal one. Hence, it is surprising that a local search algorithm yields the best known performance ratio for the metric k-median problem. In this section we describe the algorithm of Arya et al. [5] which for any value $\varepsilon > 0$, achieves a performance ratio of $(3 + 2/p)/(1 - \varepsilon n^2) \leq 3 + 2/p + \varepsilon'$ for $\varepsilon' = 10\varepsilon n^2$, if $\varepsilon \leq 1$ and $n \geq 2$. This algorithm repeatedly improves a solution by swapping p of the centers in the current solution with p facilities not in the solution, where the value of the parameter p is not larger than k.

Given a set S of at most k centers we define the cost of S, denoted as $c(S)$, as the total service costs of the clients, i.e. $c(S) = \sum_{d \in D} c(d, S)$, where $c(d, S) = \min\{c(d, f) \mid f \in S\}$ is the smallest cost of servicing d by one of the centers in S. Let ε be a constant value and $0 < \varepsilon < 1$. The algorithm of Arya et al. is the following.

Algorithm LocalSearch (p, ε)

```
1. $S^\ell \leftarrow$ an arbitrary set of $k$ centers
2. while there are sets $T \subseteq F \setminus S^\ell$ and $T' \subseteq S^\ell$ such that $|T| = |T'| \leq p$
           and $c((S^\ell \setminus T') \cup T) \leq (1 - \varepsilon)c(S^\ell)$, do
               $S^\ell \leftarrow (S^\ell \setminus T') \cup T$
3. return $S^\ell$.
```

Let S^* be an optimum solution for the k-median problem. If p and ε are constant values, a straightforward implementation of the algorithm checks in every iteration of the while loop all subsets of $F\backslash S^\ell$ and S^ℓ of size at most p. Since there are $O(n^p)$ subsets of $F \setminus S^\ell$ (and of S^ℓ) of size at most p, and since the condition $c((S^\ell\backslash T')\cup T) \le (1-\varepsilon)c(S^\ell)$ can be easily tested in $O(n)$ time, then each iteration of the while loop can be implemented to run in $O(n^{2p+1})$ time. In every iteration of the while loop the value of the solution decreases by at least a factor of $1-\varepsilon$, and so the maximum number of iterations is $\log_{\frac{1}{1-\varepsilon}} \frac{c(S_0)}{c(S^*)}$, where S_0 is the initial solution. Hence, the algorithm runs in time $O(n^{2p+1}\log(c(S_0)/c(S^*))/\log(\frac{1}{1-\varepsilon}))$.

Theorem 8. *Algorithm* LocalSearch *has a performance ratio of* $(3+2/p)/(1-\varepsilon)$.

In order to ensure that the above algorithm has polynomial running time, we must guarantee that each iteration of the **while** loop decreases the value of the current solution by at least a factor of $1-\varepsilon$. Therefore, the algorithm is not guaranteed to find a locally optimal solution. To prove Theorem 8 we first need to show some properties of the solution S^ℓ computed by the algorithm. For any sets $T \subseteq S^\ell$ and $T' \subseteq F \setminus S^\ell$ such that $|T| = |T'| \le p$,

$$c((S^\ell \setminus T) \cup T') > (1-\varepsilon)c(S^\ell). \tag{13}$$

Given a feasible solution S for the k-median problem, we denote the set of clients serviced by some subset of centers $A \subseteq S$ as $N_S(A)$. Given a center $s \in S$, the set of clients served by s is denoted as $N_S(s)$.

Given a set of centers $A \subseteq S^\ell$, we say that A *captures* a center o belonging to the optimum solution S^*, if A serves at least half of the clients served by o, or in other words, if $|N_{S^\ell}(A) \cap N_{S^*}(o)| \ge |N_{S^*}(o)|/2$. We define *capture*$(A)$ as the set of centers $o \in S^*$ captured by A. For any two sets $X, Y \subseteq S^\ell$, the following properties hold.

Lemma 12. *If X and Y are disjoint, then* capture(X) *and* capture(Y) *are disjoint. Furthermore, if $X \subseteq Y$ then* capture$(X) \subseteq$ capture(Y).

Proof. To prove the first property we show that every center $o \in S^*$ can be captured by at most one of the sets X, Y. If a center $o \in S^*$ is captured by, say, X then more than half of the clients served by o in the optimum solution are served by centers in X. Therefore, the centers in Y cannot capture o.

The second property is easy to show since every center $o \in S^*$ captured by X has more than half of its clients served also by Y in S^ℓ.

We call a center $s \in S^\ell$ *bad* if it captures at least one center in S^*, and *good* otherwise.

Lemma 13. *The solutions S^ℓ and S^* can be partitioned into sets $A_1, A_2, \ldots, A_r$ and $B_1, B_2, \ldots, B_r$, respectively, where $r-1$ is the number of bad centers in S^ℓ. This partition is such that for all $i = 1, \ldots, r-1$, $|A_i| = |B_i|$, $B_i =$*capture(A_i)*, and A_i has one bad center. Furthermore, $|A_r| = |B_r|$ and A_r has only good centers.*

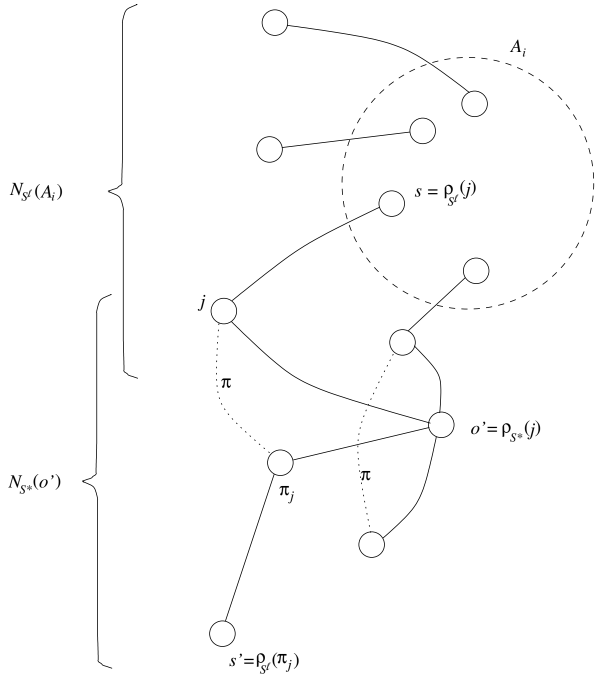

Fig. 2. Service cost of client j after reassigning clients to centers

Proof. We prove the lemma by induction on the number of bad centers. The basis is trivial because if the number of bad centers is 0 then we simply set $A_1 = S^\ell$ and $B_1 = S^*$. For the induction step, let us assume that the partitions exist when the number of bad centers in S^ℓ is b, $b \geq 0$, and prove that such a partition exists when the number of bad centers is $b+1$. Choose a bad center $s \in S^\ell$, and set $A_1 = \{s\}$. Note that $capture(A_1) \geq 1$. If $|capture(A_1)| > |A_1|$, then repeatedly add good centers to A_1 until the size of A_1 is equal to the size of $capture(A_1)$. We can always do this since every bad center in S^ℓ captures at least one center in S^* and, hence, if $|capture(A_1)| > |A_1|$ there must be at least one good center in $S^\ell \setminus A_1$.

Set $B_1 \leftarrow capture(A_1)$. To complete the proof, we note that by the induction hypothesis it is possible to partition $S^\ell \setminus A_1$ and $S^* \setminus capture(A_1)$ into sets $A_2, \ldots, A_{k+2}$ and $B_2, \ldots, B_{k+2}$ as described in the statement of the lemma. By Lemma 12, the sets B_i are disjoint, so the sets A_i and B_i are partitions of S^ℓ and S^*, respectively. Finally, since $|S^\ell| = |S^*|$, then $|A_r| = |B_r|$.

For each client j let $\rho_{S^\ell}(j)$ be the center that serves j in solution S^ℓ and let $\rho_{S^*}(j)$ be the center that serves j in S^*. Now we are ready to prove Theorem 8.

Proof of Theorem 8. Let $\{A_1, A_2, \ldots, A_r\}$, $\{B_1, B_2, \ldots, B_r\}$ be partitions of S^ℓ and S^* as described above. To compute the performance ratio of the algorithm, let us consider the following sets of swaps involving all the centers in the optimum solution S^*.

i. For each set A_i such that $|A_i| = |B_i| \leq p$, swap the centers in A_i with those in B_i. By inequality (13) we know that

$$c((S^\ell \setminus A_i) \cup B_i) > (1-\varepsilon)c(S^\ell).$$

We can bound the value of the left hand side of this inequality by reassigning the clients served by S^ℓ to centers in $(S^\ell \setminus A_i) \cup B_i$ as follows. All clients in $N_{S^*}(B_i)$ are assigned to the centers in B_i. For all other clients that are served by A_i we proceed as follows. Consider all clients j served by A_i that are served in S^* by some center $o' = \rho_{S^*}(i) \notin B_i$. Since $o' \notin B_i$, then A_i does not capture o' and, hence, $|N_{S^\ell}(A_i) \cap N_{S^*}(o')| < \frac{1}{2}|N_{S^*}(o')|$. Thus, for each one of these clients j we can associate a unique client $\pi_j \in N_{S^*}(o') \setminus N_{S^\ell}(A_i)$ (see Figure 2). Let π_j be served in S^ℓ by some center $s' = \rho_{S^\ell}(\pi_j) \notin A_i$. Then, by the triangle inequality, reassigning client j to center s' incurs a cost at most $c(\rho_{S^\ell}(\pi_j), j) \leq c(\rho_{S^\ell}(\pi_j), \pi_j) + c(\rho_{S^*}(j), \pi_j) + c(\rho_{S^*}(j), j)$. Hence,

$$\begin{aligned}
(1-\varepsilon)c(S^\ell) &< c((S^\ell \setminus A_i) \cup B_i) \\
&\leq \sum_{j \in N_{S^*}(B_i)} c(\rho_{S^*}(j), j) + \\
&\quad \sum_{\substack{j \in N_{S^\ell}(A_i) \\ j \notin N_{S^*}(B_i)}} (c(\rho_{S^\ell}(\pi_j), \pi_j) + c(\rho_{S^*}(j), \pi_j) + c(\rho_{S^*}(j), j)) + \\
&\quad \sum_{\substack{j \in D \\ j \notin N_{S^*}(B_i) \cup N_{S^\ell}(A_i)}} c(\rho_{S^\ell}(j), j) \\
&\leq \sum_{j \in D} c(\rho_{S^\ell}(j), j) + \sum_{j \in N_{S^*}(B_i)} (c(\rho_{S^*}(j), j) - c(\rho_{S^\ell}(j), j)) + \\
&\quad \sum_{j \in N_{S^\ell}(A_i)} (c(\rho_{S^\ell}(\pi_j), \pi_j) + \\
&\quad c(\rho_{S^*}(j), \pi_j) + c(\rho_{S^*}(j), j) - c(\rho_{S^\ell}(j), j)) \qquad (14)
\end{aligned}$$

ii. For each set A_i such that $|A_i| = |B_i| = q > p$, we first select a set of $q-1$ good centers from A_i. Then, we swap every center $o \in B_i$ with each one of the $q-1$ selected good centers s from A_i. Proceeding similarly as above, we can show that

$$\begin{aligned}
(1-\varepsilon)c(S^\ell) &< c((S^\ell \setminus \{s\}) \cup \{o\}) \\
&\leq \sum_{j \in D} c(\rho_{S^\ell}(j), j) + \sum_{j \in N_{S^*}(o)} (c(\rho_{S^*}(j), j) - c(\rho_{S^\ell}(j), j)) + \\
&\quad \sum_{j \in N_{S^\ell}(s)} (c(\rho_{S^\ell}(\pi_j), \pi_j) + c(\rho_{S^*}(j), \pi_j) + \\
&\quad c(\rho_{S^*}(j), j) - c(\rho_{S^\ell}(j), j))
\end{aligned}$$

By adding all inequalities for a center $o \in B_i$, and then dividing the sum by $q-1$ we get

$$\begin{aligned}(1-\varepsilon)c(S^\ell) < \sum_{j\in D} c(\rho_{S^\ell}(j),j) + \sum_{j\in N_{S^*}(o)} (c(\rho_{S^*}(j),j) - c(\rho_{S^\ell}(j),j)) +\\ \frac{1}{q-1}\sum_{j\in N_{S^\ell}(A_i)} (c(\rho_{S^\ell}(\pi_j),\pi_j) + c(\rho_{S^*}(j),\pi_j) +\\ c(\rho_{S^*}(j),j) - c(\rho_{S^\ell}(j),j))\end{aligned} \tag{15}$$

Next, we add the inequalities (15) for the q centers $o \in B_i$:

$$\begin{aligned}q(1-\varepsilon)c(S^\ell) < q\sum_{j\in D} c(\rho_{S^\ell}(j),j) + \sum_{j\in N_{S^*}(B_i)} (c(\rho_{S^*}(j),j) - c(\rho_{S^\ell}(j),j)) +\\ \frac{q}{q-1}\sum_{j\in N_{S^\ell}(A_i)} (c(\rho_{S^\ell}(\pi_j),\pi_j) + c(\rho_{S^*}(j),\pi_j) +\\ c(\rho_{S^*}(j),j) - c(\rho_{S^\ell}(j),j))\end{aligned} \tag{16}$$

Add the inequalities (14) and (16) for all sets A_i, B_i. Let r_1 be the number of pairs A_i, B_i for which $|A_i| = |B_i| \le p$. Since $\bigcup_{A_i} N_{S^\ell}(A_i) = \bigcup_{B_i} N_{S^*}(B_i) = D$, then by adding the inequalities we get

$$\begin{aligned}(r_1 + q(r-r_1))(1-\varepsilon)c(S^\ell) < (r_1 + q(r-r_1))\sum_{j\in D} c(\rho_{S^\ell}(j),j) +\\ \sum_{j\in D}(c(\rho_{S^*}(j),j) - c(\rho_{S^\ell}(j),j)) +\\ \frac{q}{q-1}\sum_{j\in D}(c(\rho_{S^\ell}(\pi_j),\pi_j) + c(\rho_{S^*}(j),\pi_j) +\\ c(\rho_{S^*}(j),j) - c(\rho_{S^\ell}(j),j))\end{aligned} \tag{17}$$

Because of the way in which the mapping π has been defined, it is not hard to see that

$$\sum_{j\in D}(c(\rho_{S^\ell}(\pi_j),\pi_j) + c(\rho_{S^*}(j),\pi_j) + c(\rho_{S^*}(j),j) - c(\rho_{S^\ell}(j),j)) \le 2c(S^*).$$

By using this last inequality in (17), we get

$$\begin{aligned}(r_1 + q(r-r_1))(1-\varepsilon)c(S^\ell) &< (r_1 + q(r-r_1))c(S^\ell) +\\ &\quad (c(S^*) - c(S^\ell)) + \frac{q}{q-1}(2c(S^*))\\ &\le \left(3+\frac{2}{p}\right)c(S^*) + (r_1 + q(r-r_1) - 1)c(S^\ell)\end{aligned}$$

The last inequality follows from $q/(q-1) \leq (p+1)/p$. Since $r_1 + q(r - r_1) < n^2$, then

$$(1 - \varepsilon n^2)c(S^\ell) < (1 - \varepsilon(r_1 + q(r - r_1)))c(S^\ell) < \left(3 + \frac{2}{p}\right) c(S^*).$$

Thus, $c(S^\ell) \leq (3 + \frac{2}{p})/(1 - \varepsilon n^2)c(S^*)$. □

Acknowledgements

I kindly thank the anonymous referees for their very accurate and insightful comments.

References

1. A. Archer, R. Rajagopalan, and D. Shmoys, Lagrangian relaxation for the k-median problem: new insights and continuity properties, *Proceedings of the 11th Annual European Symposium on Algorithms*, 2003, LNCS 2832, pp. 31–42.
2. S. Ahn, A. Cooper, G. Cornuejols, and A. Frieze, Probabilistic analysis of a relaxation for the k-median problem, *Mathematics of Operations Research*, 13, 1988, pp. 1–31.
3. M. Andrews and L. Zhang, The access network design problem, *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, 1998, pp.40–59.
4. S. Arora, P. Raghavan, and S. Rao, Approximation schemes for Euclidean k-medians and related problems, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp. 106–113.
5. V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, Local search heuristics for k-median and facility location problems, *Proceedings of the ACM Symposium on Theory of Computing*, 2001, pp. 21–29.
6. M.L. Balinski, On finding integer solutions to linear programs, *Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems*, 1966, pp. 225-248.
7. Y. Bartal, Probabilistic approximation of metric spaces and its algorithmic applications, *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*. 1996, pp. 184–193.
8. Y. Bartal, On approximating arbitrary metrics by tree metrics, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp. 161–168.
9. P.S. Bradley, U.M. Fayad, and O.L. Mangasarian, Mathematical programming for data mining: formulations and challenges, Microsoft Technical Report, 1998.
10. A.F. Bumb and W. Kern, A simple dual ascent algorithm for the multilevel facility location problem, *Proceedings of RANDOM-APPROX*, 2001, pp. 55-62.
11. M. Charikar, C. Chekuri, A. Goel, and S. Guha, Rounding via trees: deterministic approximation algorithms for group Steiner trees and k-median, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp. 114–123.
12. M. Charikar and S. Guha, Improved combinatorial algorithms for the facility location and k-median problems, *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999, pp. 378–388.

13. M. Charikar, S. Guha, E. Tardos, and D.B. Shmoys, A constant-factor approximation algorithm for the k-median problem, Proceedings of the Thirty-First Symposium on Theory of Computing, 1999, pp. 1–10.
14. J. Cheriyan and R. Ravi, *Approximation algorithms for network problems*, Lecture Notes, University of Waterloo, 1998.
15. F. Chudak and D. Williamson, Improved approximation algorithms for capacitated facility location problems, *Proceedings of the 7th Conference on Integer Programming and Combinatorial Optimization*, 1999.
16. V. Chvátal, A greedy heuristic for the set covering problem, *Mathematics of Operations Research*, 4, 1979, pp. 233–235.
17. V. Chvátal, *Linear Programming*, W.H. Freeman and Company, 1983.
18. G. Cornuejols, G.L. Nemhauser, and L.A. Wolsey, The uncapacitated facility location problem, in P. Mirchandani and R. Francis, eds. *Discrete Location Theory*, John Wiley and Sons, New York, 1990.
19. S. Guha, A. Meyerson, and K. Munagala, Hierarchical placement and network design problems, *Proceedings of the 14th Annual IEEE Symposium on Foundations of Computer Science*, 2000.
20. S. Guha, A. Meyerson, and K. Munagala, Improved combinatorial algorithms for single sink edge installation problems, Technical Report STAN-CS-TN00-96, Stanford University, 2000.
21. S. Guha, A. Meyerson, and K. Munagala, Improved algorithms for fault tolerant facility location, *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2001, pp. 636–641.
22. S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, On the placement of internet instrumentations, *Proceedings of IEEE INFOCOM*, 2000, pp. 26–30.
23. K. Jain and V.V. Vazirani, Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation, *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999. Also, *Journal of the ACM*, 48, 2001, pp. 274–296.
24. D.S. Johnson, Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, 9, 1974, pp. 256–278.
25. O. Kariv and S.L. Hakimi, An algorithmic approach to network location problems, Part II: p-medians, *SIAM Journal on Applied Mathematics*, 1979, pp. 539–560.
26. O. Kariv and S.L. Hakimi, An algorithmic approach to network location problems. II: The p-medians, *SIAM Journal on Applied Mathematics*, 1979, pp. 539–560.
27. L. Khachiyan, A polynomial algorithm for linear programming, *Doklady Akad. Nauk USSR*, 244 (5), 1979, pp. 1093–1096.
28. S. Khuller, R. Pless, and Y.J. Sussman, Fault tolerant k-center problems, *Theoretical Computer Science*, 242, 2000, pp. 237–245.
29. A.A. Kuehn and M.J. Hamburger, A heuristic program for locating warehouses, *Management Science*, 9, 1963, pp. 643–666.
30. M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems, *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 1-10. Also in *Journal of Algorithms* 37, 2000, pp. 146-188.
31. P. Krysta and R. Solis-Oba, Approximation algorithms for bounded facility location, *Journal of Combinatorial Optimization*, 5, 2001, pp. 2–16.
32. B. Li, M. Golin, G. Italiano, X. Deng, and K. Sohraby, On the optimal placement of web proxies in the internet, *Proceedings of IEEE INFOCOM* 1999, pp. 1282–1290.

33. J. Lin and J.S. Vitter, ϵ-Approximations with minimum packing constraint violation, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1992, pp. 771–782.
34. J. Lin and J.S. Vitter, Approximation algorithms for geometric median problems, *Information Processing Letters*, 44, 1992, pp. 245–249.
35. L. Lovász, On the ratio of optimal integral and fractional covers, *Discrete Mathematics*, 13, 1975, pp. 383–390.
36. N. Megiddo and K.J. Supowit, On the complexity of some common geometric location problems, *SIAM Journal on Computing*, 13, 1984, pp. 182–196.
37. M. Mahdian and M. Pál, Universal facility location, *Proceedings of the 11th Annual European Symposium on Algorithms*, 2003, LNCS 2832, pp. 409–421.
38. M. Mahdian, Y. Ye, and J. Zhang, Improved approximation algorithms for metric facility location problems, *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, 2001, LNCS 2462, pp. 229–242.
39. R. R. Mettu and C. G. Plaxton, The online median problem, *SIAM Journal on Computing*, 32, 2003, pp. 816-832.
40. J.M. Mulvey and H.L. Crowder, Cluster Analysis: an application of Lagrangian relaxation, *Management Science*, 25, 1979, pp. 329–340.
41. K. Murty, *Linear Programming*, John Wiley & Sons, 1983.
42. G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons, New York, 1990.
43. C.H. Papadimitriou, Worst case and probabilistic analysis of a geometric location problem, *SIAM Journal on Computing*, 10 (3), 1981, pp. 542–557.
44. L. Qiu, V.N. Padmanabhan, and G. Voelker, On the placement of web server replicas, *Proceedings of IEEE INFOCOM*, 2001.
45. R.Raz and S. Safra, A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP, *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*,1997, pp. 475–484.
46. A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, 1986.
47. A. Tamir, An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs, *Operations Research Letters*, 19, 1996, pp. 59–94.
48. Y. Ye, An $O(n^3L)$ potential reduction algorithm for linear programming, *Mathematical Programming*, 50, 1991, 239–258.
49. J. Zhang, Approximating the two-level facility location problem via a quasi-greedy approach, *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004, 801-810.

The Lovász-Local-Lemma and Scheduling

Anand Srivastav

Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, Christian-Albrechts-Platz 4, D-24118 Kiel, Germany
asr@numerik.uni-kiel.de
http://www.numerik.uni-kiel.de/asr

Abstract. Probabilistic methods have advanced the design of algorithms in algorithmic discrete mathematics and theoretical computer science. Many notoriously hard algorithmic problems have been solved with randomized algorithms or probabilistic methods, either optimally or in a satisfactory approximative way. One of the powerful tools in analyzing randomized approximation algorithms is the Lovász-Local-Lemma, a sieve method with many nice applications. In this article we show its impact on job shop scheduling and resource constrained scheduling.

1 Preliminaries

1.1 Graphs and Hypergraphs

We use the standard notion of graphs and hypergraphs. A graph $G = (V, E)$ is a pair consisting of a finite set V (the set of vertices or nodes) and a subset $E \subseteq \binom{V}{2}$, where $\binom{V}{2}$ denotes the set of all 2−element subsets of V. The elements of E are called edges. For a set X let $\mathcal{P}(X)$ be the powerset of X. A hypergraph or set system $\mathcal{H} = (V, \mathcal{E})$ is a pair of a finite set V and a subset $\mathcal{E} \subseteq \mathcal{P}(V)$. The elements of $\mathcal{E}$ are called hyperedges. The degree of a vertex $v \in V$ in $\mathcal{H}$, denoted by $\deg(v)$, is the number of hyperedges containing v, and $\deg(\mathcal{H}) = \max_{v\in V} \deg(v)$ is the (vertex-)degree of $\mathcal{H}$. $\mathcal{H}$ is called r-regular resp. k-uniform, if $\deg(v) = r$ for all $v \in V$ resp. $|E| = k$ for all $E \in \mathcal{E}$. For convenience we write $V = \{v_1, \dots, v_n\}$ and $\mathcal{E} = \{E_1, \dots, E_m\}$, and sometimes identify vertices and edges with their indices. The vertex-hyperedge incidence matrix of a hypergraph $\mathcal{H} = (V, \mathcal{E})$ is a matrix $A = (a_{ij}) \in \{0, 1\}^{n\times m}$, where $a_{ij} = 1$ if $v_i \in E_j$, and 0 otherwise. For a modern treatment of graph theory, we refer to the books of Berge [9], Bollobás [12], Diestel [17] and West [42].

1.2 Large Deviations

Suppose a randomized approximation algorithm for an optimization problem produces a solution with real objective function value C. Quite often it is possible to compute the expectation $\mathbb{E}[C]$, and if we are lucky, we even can prove that $\mathbb{E}[C]$ is close to the optimal value OPT, say with relative error $\epsilon > 0$: [1]

[1] ϵ can be a constant, but in many cases it is a function $\epsilon = \epsilon(|I|)$, depending on the size of the input.

$$|\mathbb{E}(C) - \text{OPT}| \leq \epsilon OPT. \tag{1}$$

To be on the critics' side, the inequality (1) does not say too much! It can happen that C is "badly" distributed around $\mathbb{E}[C]$, meaning that C may deviate significantly from $\mathbb{E}[C]$, thus destroying the nice approximation suggested by (1). If the randomized algorithm is well-designed, then of course C should stay close to $\mathbb{E}[C]$ with high probability. Such concentration phenomena are usually proved by bounding the deviation from the expectation with large deviation inequalities. Throughout this article we consider only finite probability spaces $(\Omega, \mathbb{P})$, where Ω is a finite set and $\mathbb{P}$ is a probability measure with respect to the powerset $\mathcal{P}(\Omega)$ as the sigma field. Let $u_1, \dots, u_n$ and $v_1, \dots, v_n$ be integers, and let $X_1, \dots, X_n$ be mutually independent (briefly independent) random variables, where X_j takes the values u_j or v_j with probability

$$\mathbb{P}[X_j = u_j] = p_j,\ \mathbb{P}[X_j = v_j] = 1 - p_j$$

for real valued p_j, $1 \leq j \leq n$. For $1 \leq j \leq n$ let w_j denote rational weights with

$$0 \leq w_j \leq 1$$

and let

$$\psi = \sum_{j=1}^{n} w_j X_j$$

be the weighted sum. For $u_j = 1$, $v_j = 0$, $w_j = 1$ and $p_j = p$ for all $j = 1, \dots, n$, $\psi = \sum_{j=1}^{n} X_j$ is the well-known binomially distributed random variable with mean np. The inequalities given below and their proofs can be found in the books of Alon, Spencer and Erdős [6], Habib, McDiarmid, Ramirez-Alfonsin and Reed [23], and Janson, Łuczak, Ruciński [26].

Theorem 1. (Markov Inequality) *Let $(\Omega, \mathbb{P})$ be a probability space and $X : \Omega \longrightarrow \mathbb{R}^+$ a random variable with expectation $\mathbb{E}[X] < \infty$. Then for any $\lambda \in \mathbb{R}^+$*

$$\mathbb{P}[X \geq \lambda] \leq \frac{\mathbb{E}[X]}{\lambda}.$$

A sharper bound is the well-known inequality of Chebyshev:

Theorem 2. (Chebyshev Inequality) *Let $(\Omega, \mathbb{P})$ be a probability space and $X : \Omega \longrightarrow \mathbb{R}$ a random variable with expectation $\mathbb{E}[X]$ and variance $Var[X]$. Then for any $\lambda \in \mathbb{R}^+$*

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq \lambda\sqrt{Var[X]}] \leq \frac{1}{\lambda^2}.$$

The following basic large deviation inequality is implicitly given in Chernoff [14] in the Binomial case. In explicit form it can be found in Okamoto [32]. Its generalization to arbitrary weights is due to Hoeffding [25]:

Theorem 3. (Hoeffding 1963) *Let $u_j = 1, v_j = 0, 0 \leq w_j \leq 1, 0 \leq p_j \leq 1$ for all $j = 1, \dots, n$ and let $\lambda > 0$. Then*

(a) $\mathbb{P}[\psi \geq \mathbb{E}[\psi] + \lambda] \leq \exp(-\frac{2\lambda^2}{n})$
(b) $\mathbb{P}[\psi \leq \mathbb{E}[\psi] - \lambda] \leq \exp(-\frac{2\lambda^2}{n})$.

In the literature Theorem 3 is well-known as the Chernoff bound. Another useful form is given in the following theorem (see [36] for a proof).

Theorem 4. *Let* $u_j = 1, v_j = 0, 0 \leq w_j \leq 1, 0 \leq p_j \leq 1$ *for all* $j = 1, \ldots, n$ *and let* $\mu = \mathbb{E}[\psi]$. *Then*

(i) For any $\beta > 0$, $\mathbb{P}[\psi \geq \mu(1+\beta)] \leq G(\mu, \beta)$ *where*

$$G(\mu, \beta) = \left(\frac{e^\beta}{(1+\beta)^{(1+\beta)}}\right)^\mu .$$

(ii) $\forall \mu > 0$ *and* $\forall p \in (0,1)$, *there exists* $\beta = H(\mu, p) > 0$ *such that* $\lceil \mu\beta \rceil \cdot G(\mu, \beta) \leq p$ *and*

$$H(\mu, p) = \begin{cases} \Theta\left(\sqrt{\frac{\log(p^{-1}+\mu)}{\mu}}\right) & \textit{if } \mu > \frac{\log p^{-1}}{2} \text{ ;} \\ \Theta\left(\frac{\log p^{-1}}{\mu \log(\log(p^{-1})/\mu)}\right) & \textit{otherwise .} \end{cases}$$

For random variables with zero expectation we have an inequality due to Hoeffding [25]:

Theorem 5. (Hoeffding 1963) *Let* $u_j = 1,\ v_j = -1,\ w_j = 1,\ p_j = 1/2$ *for all* $j = 1, \ldots, n$. *For* $\lambda > 0$ *we have*

(a) $\mathbb{P}[\psi > \lambda] \leq \exp(-\frac{\lambda^2}{2n})$
(b) $\mathbb{P}[\psi < -\lambda] \leq \exp(-\frac{\lambda^2}{2n})$.

For small expectations, i.e., $\mathbb{E}[\psi] \leq \frac{n}{6}$, the following inequalities due to Angluin and Valiant [7] give sharper bounds than Chernoff's inequality.

Theorem 6. (Angluin, Valiant 1979) *Let* $u_j = 1, v_j = 0, 0 \leq w_j \leq 1, 0 \leq p_j \leq 1$ *for all* $j = 1, \ldots, n$ *and let* $0 < \beta \leq 1$. *Then*

(a) $\mathbb{P}[\psi > \mathbb{E}[\psi](1+\beta)] \leq \exp(-\frac{\beta^2 \mathbb{E}[\psi]}{3})$
(b) $\mathbb{P}[\psi < \mathbb{E}[\psi](1-\beta)] \leq \exp(-\frac{\beta^2 \mathbb{E}[\psi]}{2})$.

2 The Lovász-Local-Lemma

The Lovász-Local-Lemma was introduced by Erdős and Lovász [18] in the study of 2-colorings of hypergraphs with forbidden monochromatic edges. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. Let the *degree* of an edge $E \in \mathcal{E}$ be the number of edges intersecting E and let the *edge-degree* of the hypergraph be the maximum edge-degree. As usual, the size of $\mathcal{H}$ is the number of edges. Let us assume that this size is m. A 2-coloring is a mapping from V to a set of two colors [2], say red and blue. It is convenient to represent the colors either by $\{-1, 1\}$ or $\{0, 1\}$. A 2-coloring of V is called non-monochromatic, if no hyperedge is monochromatic with respect to the coloring.

Definition 1. *A hypergraph $\mathcal{H} = (V, \mathcal{E})$ has property $\mathcal{B}$ if there is a 2-coloring of V such that no hyperedge of $\mathcal{H}$ is monochromatic.*

In 1975, Erdős and Lovász [18] showed that a k-uniform hypergraph with edge degree at most 2^{k-3} has property $\mathcal{B}$. The remarkable conclusion of this theorem is that without any assumption on the size of the hypergraph a merely local condition suffices to guarantee property $\mathcal{B}$. The key in the proof is a sieve method, today known as the Lovász-Local-Lemma (LLL), with many applications in combinatorial optimization, graph theory, and combinatorics (for example to k-coloring of real numbers, Ramsey theory [18], linear and star arboricity of graphs [2,4], acyclic colorings of graphs [5], and construction of edge disjoint paths in expander graphs [13]).

Theorem 7. (Lovász-Local-Lemma; Symmetric Case) *Let $A_1, \ldots, A_n$ be events with $\mathbb{P}[A_i] \le p$ for all i. Suppose that each event A_i is mutually independent of all but at most $d > 1$ other events A_j. If $ep(d+1) \le 1$ then*

$$\mathbb{P}[\bigwedge_{i=1}^{n} A_i^c] \ge \left(1 - \frac{1}{d+1}\right)^n .$$

Proof. Suppose, we have already shown that for each $S \subset \{1, \ldots, n\}$ and any $i \in \{1, \ldots, n\} \setminus S$,

$$\mathbb{P}[A_i \mid \bigwedge_{j \in S} A_j^c] \le \frac{1}{d+1}. \tag{2}$$

Then

$$\mathbb{P}[\bigwedge_{i=1}^{n} A_i^c] = \prod_{i=1}^{n} (1 - \mathbb{P}[A_i \mid \bigwedge_{j=1}^{i-1} A_j^c]) \ge \left(1 - \frac{1}{d+1}\right)^n ,$$

as claimed. We will prove (2) by induction on $|S|$. If $S = \emptyset$, (2) follows from the fact that $p \le \frac{1}{e(d+1)} < \frac{1}{d+1}$. Now assume that (2) holds for all $S' \subset S$. For the induction step we fix $i \in S$ and let $T \subset S$ consist of the indices of all events that are not mutually independent of A_i. We renumber the events such that $T = \{1, \ldots, t\}$, $t \le d$. Observe that

$$\mathbb{P}[A_i \mid \bigwedge_{j \in S} A_j^c] = \frac{\mathbb{P}[A_i \wedge \bigwedge_{j \in T} A_j^c \mid \bigwedge_{j \in S \setminus T} A_j^c]}{\mathbb{P}[\bigwedge_{j \in T} A_j^c \mid \bigwedge_{j \in S \setminus T} A_j^c]}.$$

[2] Our notion of a coloring does not require that adjacent vertices have different colors.

We can bound the numerator from above by

$$\mathbb{P}[A_i \mid \bigwedge_{j\in S\setminus T} A_j^c],$$

and since A_i is mutually independent of the events $\{A_j \mid j \in S \setminus T\}$ we have

$$\mathbb{P}[A_i| \bigwedge_{j\in S\setminus T} A_j^c] = \mathbb{P}[A_i] = p.$$

The denominator satisfies

$$\mathbb{P}[\bigwedge_{j\in T} A_j^c \mid \bigwedge_{j\in S\setminus T} A_j^c] = \prod_{j=1}^{t}(1 - \mathbb{P}[A_j \mid \bigwedge_{k=1}^{j-1} A_k^c \wedge \bigwedge_{l\in S\setminus T} A_l^c]),$$

which is at least $\prod_{j=1}^{t}\left(1-\frac{1}{d+1}\right) = \left(1-\frac{1}{d+1}\right)^t$ by the induction assumption. Hence we obtain

$$\mathbb{P}[A_i \mid \bigwedge_{j\in S} A_i^c] \le \frac{p}{\left(1-\frac{1}{d+1}\right)^t} \le \frac{p}{\left(1-\frac{1}{d+1}\right)^d} \le \mathrm{e}p \le \frac{1}{d+1},$$

completing the induction. □

As a very simple application which demonstrates the strength of the LLL we derive a lower bound on the van der Waerden numbers $W(k)$. Recall that $W(k)$ denotes the minimal integer n so that every 2-coloring of the set $\{1,\dots,n\}$ contains a monochromatic arithmetic progression of length k. Let us first prove a lower bound from a standard probabilistic argument. We consider a random 2-coloring where each $i \in \{1,\dots,n\}$ has probability $1/2$ of being colored red or blue, independently of all $j \neq i$. For each arithmetic progression S of length k let A_S be the event that S is monochromatic. If the event $\bigwedge_S A_S^c$ has nonzero probability we know that there exists a 2-coloring of $\{1,\dots,n\}$ containing no monochromatic k-term arithmetic progression. Hence $W(k) > n$ as long as $\mathbb{P}[\bigwedge_S A_S^c] > 0$. Clearly, $\mathbb{P}[A_S] = 2^{1-k}$, and since there are $\frac{n^2}{2k}(1-o(1))$ k-term arithmetic progressions in $\{1,\dots,n\}$, [3] we have

$$\mathbb{P}[\bigwedge_S A_S^c] = 1 - \mathbb{P}[\bigvee_S A_S] \ge 1 - \sum_S \mathbb{P}[A_S] = 1 - \frac{n^2}{2k}(1-o(1)) \cdot 2^{1-k}.$$

The last term is positive as long as $n \le \sqrt{k}\cdot 2^{k/2}(1+o(1))$, giving $W(k) = \Omega(\sqrt{k}2^{k/2})$. The LLL yields a considerable improvement.

[3] A k-term arithmetic progression in $\{1,\dots,n\}$ can start with any $i \in \{1,\dots,n-k+1\}$ and for a fixed start term i we have $\frac{n-i+1}{k}$ possible distances between consecutive terms. Since start term and distance uniquely determine an arithmetic progression, their overall number amounts to $\sum_{i=1}^{n-k+1}\frac{n-i+1}{k} = \frac{n^2}{2k}(1-o(1))$.

Corollary 1.

$$W(k) = \Omega\left(\frac{2^{k-1}}{k}\right)$$

Proof. Consider a random 2-coloring and events A_S as above. Since A_S is mutually independent of all $A_{S'}$ with $S \cap S' = \emptyset$, we can apply the LLL with $d = \frac{nk^2}{k-1}$ (for S' with $S \cap S' \neq \emptyset$ there are at most k possibilities for an element $s \in S \cap S'$; we have at most k possibilities for the position of s within S', and at most $\frac{n}{k-1}$ choices for the step size of the arithmetic progression S'). Hence, $\bigwedge_S A_S^c$ has non-zero probability as long as $e \cdot 2^{1-k}(\frac{nk^2}{k-1} + 1) \leq 1$, which implies the claim. □

Let $A_1, \ldots, An$ be events in a probability space. A graph on $\{A_1, \ldots, A_n\}$ is called *dependency graph* of the events $A_1, \ldots, A_n$ if, for all i, the event A_i is mutually independent of $\{A_j \mid \{i,j\} \notin E(G)\}$. In a similar manner the *dependency graph $\mathcal{D}(\mathcal{H})$ of a hypergraph* $\mathcal{H} = (X, \mathcal{E})$ is defined.

Definition 2. *Let $\mathcal{H} = (X, \mathcal{E})$ be a hypergraph. The* dependency graph $\mathcal{D}(\mathcal{H})$ *is the graph with $\mathcal{E}$ as the vertex set, where $E, F \in \mathcal{E}$ form an edge $\{E, F\}$ iff $E \cap F \neq \emptyset$.*

The notion of dependency graphs will be useful later. For property $\mathcal{B}$ we get:

Corollary 2. *A hypergraph $\mathcal{H} = (V, \mathcal{E})$ with maximum edge degree* $0 < d \leq 2^{k-3}$ *in which each edge has at least $k \geq 5$ vertices has property $\mathcal{B}$.*

Proof. We randomly and independently color the nodes of $\mathcal{H}$ with two colors, say red and blue, with probability 1/2. Let us denote this random 2-coloring by χ. For an edge $E \in \mathcal{E}$ let A_E be the event that E is monochromatic with respect to χ. Put $p = 2^{-k+1}$. We have $\mathbb{P}[A_E] = 2 \cdot 2^{-|E|} \leq p$. Furthermore $ep(d+1) \leq e2^{-k+1}(2^{k-3}+1) \leq \frac{5e}{16} < 1$ for $k \geq 5$. By the Local-Lemma (Theorem 7)

$$\mathbb{P}\left[\bigwedge_{E \in \mathcal{E}} A_E^c\right] \geq (1 - \frac{1}{d+1})^m > 0, \tag{3}$$

where $m = |\mathcal{E}|$. Hence with non-zero probability, χ is a non-monochromatic 2-coloring, and $\mathcal{H}$ has property $\mathcal{B}$. □

According to (3), it may happen that the probability of χ being non-monochromatic is exponentially small. Thus an attempt to find a non-monochromatic 2-coloring is in the words of Beck "the search for a needle in a haystack". For arbitrary hypergraphs this is even hopeless, since the problem to decide whether a hypergraph has property $\mathcal{B}$ is $\mathcal{NP}$-complete (Lovász [18]). In 1991 Beck [8] gave the first algorithmic polynomial-time search version of the Local-Lemma. We will present Beck's method, its extensions and applications in Section 7.

Now we are ready to discuss some applications of the Lovász-Local-Lemma to integer programming resp. scheduling due to Srinivasan [36], resp. Leighton, Maggs and Rao [27] and Ahuja, Srivastav [1].

3 Basic Randomized Rounding – A Warming-Up Exercise

Many times a combinatorial optimization problem can be formulated as an integer linear program (ILP). Dropping the integrality constraints, we obtain a linear programming relaxation (LP) which can be solved in polynomial-time. The LP-optimum, also called *fractional optimum*, does not only provide a bound on the integral optimum, but it often can serve as a basis for finding an integral solution that is "nearly" optimal. The key idea is to "round" each variable x_i^* of a fractional optimal solution up or down, where the probability of rounding up is given by the non-integral part $x_i^* - \lfloor x_i^* \rfloor$. This guarantees that – in expectation – the rounded solution equals the fractional optimum. We would like to ensure that the deviation from the expected value is small. Let us demonstrate this "randomized rounding"[4] technique by considering the problem of finding a maximum independent set in a k-uniform hypergraph.

Definition 3. *Let $\mathcal{H} = (V, \mathcal{E})$ be a k-uniform hypergraph. A subset S of V is called* independent *if no edge is entirely contained in S, that is if $\mathcal{P}(S) \cap \mathcal{E} = \emptyset$.*

Recall, that the vertex-hyperedge incidence matrix $A = (a_{ij})_{\substack{1 \le i \le n \\ 1 \le j \le m}}$ of a k-uniform hypergraph $\mathcal{H} = (V, \mathcal{E})$ with n vertices $v_1, \dots, v_n$ and m edges $E_1, \dots, E_m$ is defined by

$$a_{ij} = \begin{cases} 1, & \text{if } v_i \in E_j \\ 0 & \text{otherwise.} \end{cases}$$

With the help of n zero/one variables x_i indicating whether or not the vertices v_i are chosen for the independent set, we can characterize an independent set in $\mathcal{H}$ of maximum cardinality as the solution to the following integer linear program:

$$\max \sum_{i=1}^{n} x_i$$

subject to

$$\sum_{i=1}^{n} a_{ij} x_i \le k - 1 \text{ for all } j \in \{1, \dots m\}$$
$$x_i \in \{0, 1\} \text{ for all } i \in \{1, \dots n\}.$$

If we relax the condition "$x_i \in \{0, 1\}$" to "$x_i \in [0, 1]$" for all $i \in \{1, \dots, n\}$, the problem can be solved in polynomial-time. Let $x_1{}^*, \dots, x_n{}^*$ be a solution and denote by OPT* the value of the objective function. Suppose, we fix $\varepsilon > 0$ and independently round each $x_i{}^*$ to 1 or 0 with probability $(1 - \frac{\varepsilon}{2}) x_i{}^*$ resp. $1 - (1 - \frac{\varepsilon}{2}) x_i{}^*$. We get a vector $(x_1, \dots, x_n) \in \{0, 1\}^n$ which is a candidate for

[4] In its concise form, the randomized rounding method in integer programming was introduced by Raghavan and Thompson [33].

an independent set in $\mathcal{H}$. For every i let X_i be the random variable indicating if x_i^* has been rounded to 1:

$$X_i = \begin{cases} 1, & \text{if } x_i = 1 \\ 0 & \text{otherwise.} \end{cases}$$

If the events

$$B_0 : \sum_{i=1}^{n} X_i \geq (1-\varepsilon)\mathrm{OPT}^*$$

$$\text{and } \ B_j : \sum_{i=1}^{n} a_{ij} X_i \leq k-1, \ j \in \{1, \dots m\}$$

simultaneously happen with positive probability, we know that $\mathcal{H}$ contains an independent set of size at least $(1-\varepsilon)\mathrm{OPT}^*$. On account of randomized rounding, the x_i have the nice property that their expected value is x_i^* (scaled down by $\left(1-\frac{\varepsilon}{2}\right)$ to boost the probability for the "good" events B_j). We can now use the Angluin-Valiant inequality (Theorem 2.6) to bound the probability of the "bad" events B_j^c. With $\beta := \frac{\varepsilon}{2-\varepsilon}$ we obtain

$$\mathbb{P}[B_j^c] \leq \mathrm{e}^{-\frac{\varepsilon^2(k-1)}{12-6\varepsilon}} \text{ for all } j \in \{0, \dots, m\}.$$

Hence

$$\mathbb{P}\Big(\bigwedge_{j=0}^{m} B_j\Big) \geq 1 - (m+1) \cdot \mathrm{e}^{-\frac{\varepsilon^2(k-1)}{12-6\varepsilon}},$$

which is at least 1/2 for $k \geq \frac{12}{\varepsilon^2}\ln(2(m+1))$. So we have proved the following theorem.

Theorem 8. *For a k-uniform hypergraph with $k > \frac{12}{\varepsilon^2}\ln(2(m+1))$ we can find an independent set of size $(1-\varepsilon)OPT^*$, where OPT^* is the optimal value of the relaxed problem formulation, in randomized polynomial-time with probability at least 1/2.*

Note that an independent set of the claimed size can be constructed in deterministic polynomial-time with the method of conditional expectations [6].

Bibliography and Remarks: We can get rid of the condition on k (on cost of a weaker approximation guarantee) by taking advantage of positive correlations among the events B_j, which can be stated by the inequality

$$\mathbb{P}\Big[\bigwedge_{j=0}^{m} B_j\Big] \geq 1 - \prod_{j=1}^{m}(1 - \mathbb{P}[B_j^c]) + \mathbb{P}[B_0^c].$$

Srinivasan's [37] analysis of randomized rounding for integer packing and covering programs using positive correlations yields a deterministic polynomial-time algorithm for computing an independent set of size

$$\Omega\left(\mathrm{OPT}^* \cdot \sqrt[k]{\frac{\mathrm{OPT}^*}{m}}\right).$$

By applying Janson's inequality and an extended version of the Lovász-Local-Lemma, Srinivasan [36] obtained a randomized approximation guarantee of

$$\Omega\left(\frac{\mathrm{OPT}^*}{\sqrt[k]{\deg(\mathcal{H})}}\right).$$

The best lower bound on the maximum size $\alpha(\mathcal{H})$ of an independent set in terms of n, m and the *average degree* $t^{k-1} := \frac{km}{n}$ is

$$\alpha(\mathcal{H}) \geq \left(1 - \frac{1}{k}\right)\frac{n}{t}.$$

The proof relies on the analysis of a simple random sampling strategy of Alon and Spencer [6]. For *uncrowded* hypergraphs (i.e., hypergraphs containing no cycles of length 2,3 or 4) Ajtai, Komlós, Pintz, Spencer, and Szémeredi were able to improve this bound by a factor of $(\ln t)^{1/(k-1)}$. Polynomial-time derandomization of this result has been given by Fundia [20] and Bertram-Kretzberg and Lefmann [11].

4 Job Shop Scheduling

An instance of the shop scheduling problem consists of

- n jobs $J_1, \ldots, J_n$
- m machines $M_1, \ldots, M_m$
- K operations $O_1, \ldots, O_K$.

Each operation O_k belongs to some job J_j and must be processed on a specific machine M_i. The processing of the operations cannot overlap in time (*non-interference condition*).

Open shop is a shop problem in which the operations of a job can be processed in any order. In a *job shop* problem the operations must be processed in a job-dependent order. A *flow shop* problem is a special job shop problem in which each job has exactly m operations, one for every machine, and the order in which the operations have to be processed is the same for all jobs. The following notations are needed:

- p_k is the processing time of operation O_k, and $p_{\max} := \max_k p_k$.
- P_j is the total processing time of job J_j, and $P_{max} := \max_j P_j$ is the maximum job-load.
- Π_i is the total processing time that is spent on machine M_i and $\Pi_{max} := \max_i \Pi_i$ is the maximum machine-load.
- μ_j is the number of operations for job J_j, and $\mu := \max_j \mu_j$.

The goal is to minimize the makespan of a feasible schedule, defined as the time at which all operations are completely processed. All of the described shop problems are $\mathcal{NP}-$hard. For a detailed discussion of approximation algorithms for scheduling we refer to the survey article of Hall [24]. Here we only review the impact of the Lovász-Local-Lemma on the job shop problem. Let us consider a job shop problem in which all operations have unit length and each job has at most one operation per machine. One of the most remarkable and surprising results for this job shop problem was proved by Leighton, Maggs and Rao [27].

Theorem 9. (Leighton, Maggs, Rao 1994) *There exists a schedule whose makespan is at most $\mathcal{O}(\Pi_{max} + P_{max})$.*

The proof relies on the Lovász-Local-Lemma (Theorem 7). One important concept is the notion of *random delays*: let us schedule the jobs in a greedy manner, where every job starts at time zero and is processed until completion. We expect such a schedule to be infeasible, violating the non-interference condition. Now we choose for every job's greedy schedule a time $t \in \{0, 1, 2, \ldots, \Delta\}$ and delay the job's starting time by t. Let us call such a schedule a greedy *pseudo-schedule* with $\{0, \Delta\}$-delay. Note that the makespan of such a delayed pseudo-schedule is at most $P_{max} + \Delta$. Randomization comes in by selecting t *randomly* from $\{0, \ldots, \Delta\}$.
The analysis of such a scheduling strategy consists of two steps :

1.) First we show with the Lovász-Local-Lemma that a pseudo-schedule with $\{0, O(\Pi_{\max})\}$ delay exists such that for each machine, the number of jobs performed per time interval of length $\Theta(\log \Pi_{\max})$ is at most one. This is the key step. The makespan of the pseudo-schedule is $O(P_{\max} + \Pi_{\max})$.
2.) In the second step we transform the pseudo-schedule into a feasible schedule of makespan $O(2^{O(\log^*(\Pi_{\max}+P_{\max}))}(P_{\max} + \Pi_{\max}))$.
More technical work is needed to conceal the $O(\log^*(\Pi_{\max} + P_{\max}))$ factor and to obtain Theorem 9.

Here we concentrate on step 1. For simplicity, we assume $\mu = 1$ and $\Pi_{max} \geq P_{max}$. Let us consider a time interval I of length T in a random $\{0, \alpha\Pi_{max}\}$-delayed schedule and let C be the largest number of operations performed on any of the machines over I. We say that the *relative congestion* of that time interval is C/T. We would like to choose T in a way that:

(a) $C \leq T$,
(b) T is small,
(c) α is a constant.

Of course, if α is large we do not expect to have problems in order to meet (a) and (b) simultaneously. The Local-Lemma proves that α and T can be kept *simultaneously* small proving 1).

Lemma 1. *There exists a greedy $\{0, 2\Pi_{\max}\}$-delayed pseudo-schedule such that the relative congestion of any time interval of length $T \geq 36\ln(2\Pi_{\max})$ is at most 1.*

Proof. The "bad" event is that the relative congestion is greater than 1. This means that more than T operations get assigned to machine M_i, for some interval I of length $T \geq 36\ln(2\Pi_{\max})$. Let A_i be this bad event. Thus the good event is $\bigwedge_{i=1}^{m} A_i^c$ and we wish to bound its probability away from 0.

Claim: (i) Let d be the dependency among the events $A_1, \ldots, A_m$. Then

$$d \leq P_{\max} \Pi_{\max}.$$

(ii) $\mathbb{P}[A_i] \leq \frac{1}{21} \frac{1}{\Pi_{\max}^4}$

The claim immediately completes the proof: the Local-Lemma condition reads

$$ep(d+1) \underset{(i),(ii)}{\leq} \frac{e}{21 \cdot \Pi_{\max}^4} \cdot (P_{\max}\Pi_{\max} + 1) \leq \frac{\Pi_{\max}^2 + 1}{7\Pi_{\max}^4} \leq 1,$$

since $\Pi_{\max} \geq 1$, and the Lemma is proven.

It remains to prove the Claim.

To (i): We consider a machine M_i and the event A_i. For any machine M_l that does not have a job in common with machine M_i, A_l is independent of A_i. The number of machines that do have a job in common with M_i is bounded by the number of jobs running on M_i (which is at most $\Pi_{\max}$) times the number of operations per job (which is at most $P_{\max}$), so $d \leq \Pi_{\max} P_{\max}$.

To (ii): We consider an arbitrary, but fixed time interval of length T,

$$36 \ln(2\Pi_{\max}) \leq T \leq \Pi_{\max},$$

and denote by q the probability that more than T operations are performed in this time interval. Let k be the number of operations that use machine M_i. Clearly $k \leq \Pi_{\max}$. By assumption each of these operations belongs to a different job. Thus, assigning to each of these operations a 0/1 random variable $X_j, j = 1, \ldots, k$, which is 1 iff operation j falls in the time interval I and 0 otherwise, the variables X_j are independent! Moreover,

$$\mathbb{P}[X_j = 1] \;\leq\; \frac{T}{2\Pi_{\max}} \;=:\; p^*$$

for all j. Now, $X := \sum_{j=1}^{k} X_j$ is the number of operations falling into I, and

$$\mathbb{E}[X] \;\leq\; kp^* \;\leq\; \frac{T}{2\Pi_{\max}} \cdot \Pi_{\max} \;=\; \frac{T}{2}$$

W.l.o.g. we may assume that $\mathbb{E}[X] = \frac{T}{2}$. We get:

$$\begin{aligned}
q &= \mathbb{P}[X > T] \\
&= \mathbb{P}\left[X > \frac{T}{2}(1+\beta)\right] \quad \text{(with } \beta = 1\text{)} \\
&= \mathbb{P}[X > \mathbb{E}[X](1+\beta)] \\
&\leq e^{-\dfrac{\mathbb{E}[X] \cdot \beta^2}{3}} \quad \text{(with Theorem 6, Angluin-Valiant inequality)} \\
&= e^{-\frac{T}{6}} \\
&\leq \left(\frac{1}{2 \cdot \Pi_{\max}}\right)^6 \quad \text{(since } T \geq 36 \ln(2 \cdot \Pi_{\max})\text{)}.
\end{aligned} \tag{4}$$

$P_{\max} + 2\Pi_{\max} \leq 3\Pi_{\max}$ is the absolute upper bound on the makespan. The number of time intervals of a certain size T, $T \geq 1$, thus is at most $3\Pi_{\max}$. Now with (4):

$$\mathbb{P}[A_i] \leq \sum_{T=1}^{\Pi_{\max}} (3\Pi_{\max})\, q \leq \frac{3\Pi_{\max}^2}{(2 \cdot \Pi_{\max})^6} \leq \frac{1}{21} \cdot \frac{1}{\Pi_{\max}^4}. \qquad \square$$

For general job-shop scheduling Shmoys, Stein and Wein [35] gave a deterministic polynomial-time algorithm which delivers a schedule of makespan $\mathcal{O}(\rho \cdot (\Pi_{\max} + P_{\max}))$ with

$$\rho = \frac{\log(m\mu)}{\log\log(m\mu)} \log(\min\{m\mu, p_{\max}\}).$$

The presently best algorithm for job-shop scheduling is due to Goldberg, Paterson, Srinivasan, Sweedyk [22], improving the above by a factor of $1/\log\log(m\mu)$.

Theorem 10. (Goldberg, Paterson, Srinivasan, Sweedyk 2001) *There is a derandomized ρ-approximation for job-shop scheduling with*

$$\rho = \frac{\log(m\mu)}{\log\log(m\mu)} \left\lceil \frac{\log(\min\{m\mu, p_{\max}\})}{\log\log(m\mu)} \right\rceil.$$

For $m = n$ it can be parallelized.

Bibliography and Remarks: Czumai and Scheideler [15,16] have provided an approach to design polynomial-time algorithms for problems that require the Lovász Local Lemma in its general unsymmetric form. In particular, they proved for the *acyclic* job shop problem, where no job has more than one operation on the same machine, that for every constant $\varepsilon > 0$, a legal schedule can be computed in polynomial-time with makespan bounded by

$$O\left((\Pi_{\max} + P_{\max}(\log\log p_{\max})^{1+\varepsilon}\right) \cdot \frac{\log p_{\max}}{\log \min\{p_{\max}, \frac{\Pi_{\max}}{P_{\max}} + (\log\log p_{\max})^{1+\varepsilon}\}}.$$

5 Resource Constrained Scheduling

5.1 Problem and History

An instance of resource constrained scheduling consists of:

- a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of independent jobs. Each job J_j needs one time unit for its completion and cannot be scheduled before its start time $r_j \in \mathbb{N}_0$.
- a set $\mathcal{P} = \{P_1, \ldots, P_m\}$ of identical processors. Each job needs one processor.
- a set $\mathcal{R} = \{R_1, \ldots, R_s, R_{s+1}\}$ of limited resources. This means that at any time all resources are available, but the available amount per time step for each resource R_i is bounded by $b_i \in \mathbb{N}$, $i = 1, \ldots, s$. The set of processors is considered as resource R_{s+1} with bound $b_{s+1} = m$.[5]

[5] The problem remains interesting if the processor constraint R_{s+1} is omitted. We will consider this variant in Section 5.3.

– For $i = 1, \ldots, s+1$, $j = 1 \ldots, n$ let $R_i(j)$ be 0/1 resource requirements indicating whether or not J_j needs resource R_i during its processing time. For a job $J_j \in \mathcal{J}$ and a time $z \in \mathbb{N}_0$ let x_{jz} be the 0/1 variable which is 1 iff job J_j is scheduled at time z.
– Given a valid schedule let C_{max} be the latest completion time defined by $C_{max} = \max\{z \mid x_{jz} > 0,\ j = 1, \ldots, n\}$.

The combinatorial optimization problem is to find a schedule, that is a 0/1 assignment for all variables x_{jz}, subject to the start time, processor and resource constraints such that $\sum_{z \in \mathbb{N}_0} x_{jz} = 1$ for all jobs J_j and C_{max} is minimum. Let C_{opt} denote this minimum.[6]

A fractional schedule is an assignment of a rational number in the closed interval $[0,1]$ to each x_{jz} subject to the start times, processor and resource constraints so that $\sum_{z \in \mathbb{N}_0} x_{jz} = 1$ for all jobs J_j and C_{max} is minimum. Let C denote this minimum. We call C_{opt} the (integral) optimal and C the fractional optimal schedule.

The integral problem is $\mathcal{NP}$-hard even if $r_j = 0$ for all $j = 1, \ldots, n$, $s = 1$ and $m = 3$ [21], while the fractional problem can be solved by linear programming in polynomial-time. Polynomial-time approximation algorithms for resource constrained scheduling with zero start times (problem class $P|\text{res} \cdots, r_j = 0, p_j = 1|C_{\max}$) are due to Garey, Graham, Johnson, Yao [21] and Röck and Schmidt [34]. Garey et al. constructed with the First-Fit-Decreasing heuristic a schedule of length C_{FFD} which asymptotically is an $(s + \frac{1}{3})$-factor approximation, i.e., there is a non-negative integer C_0 such that $C_{FFD} \leq C_{opt}(s + \frac{1}{3})$ for all instances with $C_{opt} \geq C_0$. De la Vega and Lueker [41] improved this result presenting for every $\epsilon > 0$ a linear-time algorithm which achieves an asymptotic approximation factor of $s + \epsilon$. Röck and Schmidt showed, employing the polynomial-time solvability of the simpler problem with two processors, [7] an $\lceil \frac{m}{2} \rceil$-factor polynomial-time approximation algorithm. Thus for problems with small optimal schedules or many resource constraints resp. processors these algorithms have a weak performance. Note that all of these results are based on the assumption that the start-times of all jobs are zero. For example, Röck and Schmidt's algorithm cannot be used, when non-zero start-times are given. [8]

5.2 The First Approximation

With randomized rounding and derandomization an approximation guarantee independent of the number of resources can by proved [40]:

[6] According to the standard notation of scheduling problems the integral problem can be formalized as $P|\text{res} \cdot\cdot 1, r_j, p_j = 1|C_{\max}$. This notation means: the number of identical processors is part of the input ($P|$), resources are involved (res), the number of resources and the amount of every resource are part of the input (res $\cdot \cdot$), every job needs at most 1 unit of a resource (res $\cdot \cdot 1$), start times are involved (r_j), the processing time of all jobs is equal to 1 ($p_j = 1$) and the optimization problem is to schedule the jobs as soon as possible ($|C_{max}$).

[7] Problem class $P2|\text{res} \cdot \cdot \cdot, r_j = 0, p_j = 1|C_{\max}$.

[8] This is due to the $\mathcal{NP}$-completeness of the problem $P2|\text{res} \cdot \cdot 1, r_j, p_j = 1|C_{\max}$.

Theorem 11. (Srivastav, Stangier 1997) *Let $\epsilon > 0$ with $(1/\varepsilon) \in \mathbb{N}$. For the resource constrained scheduling problem with given start times a valid integral schedule of size at most $\lceil (1+\epsilon)C \rceil$ can be found in polynomial-time, provided that $m, b_i \geq \frac{3(1+\epsilon)}{\epsilon^2} \lceil \log(8Cs) \rceil$ for all $i = 1, \ldots, s$.*

Here we give the proof of the randomized approximation guarantee, and its improvement by the LLL. For the derandomization we have to refer to [39]. The randomized algorithm behind Theorem 11 consists of 3 steps.

First, we compute a fractional schedule and the number C.
Secondly, the fractional schedule is enlarged to $\lceil (1+\varepsilon) \rceil C$. In the enlarged schedule a fraction $\tilde{x}_{jz}$ of job j is assigned to time z for all j and z.
Finally, in the enlarged schedule each job is independently assigned a random start time z with probability roughly proportional to $\tilde{x}_{jz}$. The $\tilde{x}_{jz}$ are computed with the following algorithm.

Algorithm RandomSchedule

Step 1: Let us assume that $C_{\text{opt}} \leq n$ (this assumption can be made w.l.o.g.). Start with an integer $\widetilde{C} \leq n$ and check whether the LP

$$\begin{array}{rl} \sum_{j=1}^{n} R_i(j) x_{jz} \leq b_i & \forall\, R_i \in \mathcal{R}, \\ & z \in \{1, \ldots, n\} \\ \sum_{z=1}^{n} x_{jz} = 1 & \forall\, J_j \in \mathcal{J} \\ x_{jz} = 0 & \forall\, J_j \in \mathcal{J}, z < r_j \text{ and} \\ & \forall\, J_j \in \mathcal{J}, z > \widetilde{C} \\ x_{jz} \in [0,1] & \forall\, J_j \in \mathcal{J}\ \forall z \in \{1, \ldots, n\} \end{array} \tag{5}$$

has a solution. Using binary search we can find C along with fractional assignments $(\widetilde{x}_{jz})$ by solving at most $\log n$ such LPs. Hence C can be computed in polynomial-time with standard polynomial-time LP algorithms.

Step 2: Consider the time interval $\{1, \ldots, \lceil (1+\epsilon)C \rceil\}$ and put

$$\delta = \frac{1}{1+\epsilon} \text{ and } \alpha = \frac{\epsilon\delta}{\lceil \epsilon C \rceil}.$$

Set

$$\widehat{x}_{jl} := \begin{cases} \delta \widetilde{x}_{jl} & \text{for} \quad l \in \{1, \ldots, C\} \\ \sum_{t=1}^{C} \alpha \widetilde{x}_{jt} & \text{for} \quad l \in \{C+1, \ldots, C + \lceil \varepsilon C \rceil\}. \end{cases}$$

Step 3:

Schedule the jobs at times selected by the following randomized procedure:

(a) Cast n mutually independent dice each having $N = \lceil (1+\varepsilon)C \rceil$ faces where the z-th face of the j-th die corresponding to job j appears with probability $\widehat{x}_{jz}$. (The faces stand for the scheduling times)
(b) For each $j \in \{1, \ldots, n\}$ schedule the job J_j at the time selected in (a).

It is straightforward to prove:

Lemma 2. *The variables $\widehat{x}_{jl}$ define a valid fractional schedule with makespan $\lceil (1+\varepsilon)C \rceil$.*

Proof (of Theorem 11 - Randomized Version).

Let $N = \lceil (1+\varepsilon)C \rceil$. The randomized algorithm RandomSchedule casts for every job J_j independently a dice with N faces, where the z-th face appears with probability $\widehat{x}_{jz}$ for $z \in \{1, \dots, \lceil (1+\epsilon)C \rceil\}$.

For each pair (j, z), $j \in \{1, \dots, n\}$ and $z \in \{1, \dots, N\}$ let X_{jz} be the 0/1 random variable which is 1, if the j-th dice assigns time z to job J_j and 0 otherwise. Then by definition $\mathbb{P}[X_{jz} = 1] = \widehat{x}_{jz}$.

Let A_{iz} be the event that at a time $z \in \{1, \dots, N\}$ the i-th resource constraint b_i is satisfied:

$$\text{“}\sum_{j=1}^{n} R_i(j) X_{jz} \leq b_i\text{”}.$$

The sum of units of resource R_i used at time z is a sum of independent Bernoulli-trials. Hence we can apply the Angluin-Valiant-inequality (Theorem 6):

$$\begin{aligned}\mathbb{P}[A_{iz}^c] &= \mathbb{P}\left[\sum_{j=1}^{n} R_i(j) X_{jz} > b_i\right] = \mathbb{P}\left[\sum_{j=1}^{n} R_i(j) X_{jz} > (1+\beta)\delta b_i\right] \\ &\leq \exp\left(-\frac{\beta^2 \delta b_i}{3}\right) \leq \frac{1}{4C(s+1)},\end{aligned}$$

where $\delta = \frac{1}{1+\varepsilon}$ and $\beta = \frac{1-\delta}{\delta} = \varepsilon$. The last inequality follows from the resource and processor bounds $b_i \geq \frac{3(1+\epsilon)}{\epsilon^2} \lceil \log(8Cs) \rceil$ for all i. Now, the probability that any of the events A_{iz}^c hold is at most the sum of their probability bounds. In view of the estimate above this is at most 1/2. Thus with probability at least 1/2 at any time no resource constraint is violated. □

Interestingly the approximation guarantee of Theorem 11 is best possible:

Theorem 12. (Srivastav, Stangier 1997) *Under the assumption that there exists a fractional schedule of size $C \geq 3$, and an integral schedule of size $C+1$ (C fixed), $b_i = \Omega(\log(Cs))$, $R_i(j) \in \{0, 1\}$ for all $i \in \{1, \dots, s\}$ and all $j \in \{1, \dots, n\}$ it is $\mathcal{NP}$-complete to decide whether or not there exists an integral schedule of size C.*

Proof. We give the basic argument for $b_i = 1$ for all i (the main work is its extension to large bounds $b_i = \Omega(\log(Cs))$). We use a reduction to the $\mathcal{NP}$-complete problem of determining the chromatic index of a graph. Let $G = (V, E)$ be a graph with $|V| = s$, $|E| = m$ and $\deg(v) \leq \Delta$ for all $v \in V$. We construct an instance of resource constrained scheduling as follows. Introduce for every edge $e \in E$ exactly one job J_e and consider $m = |E|$ identical processors. For every node $v \in V$ define a resource R_v with bound 1 and resource/job requirements

$$R_v(e) = \begin{cases} 1 \text{ if } & v \in e \\ 0 \text{ if } & v \notin e. \end{cases}$$

It is straightforward to verify that there exists an edge coloring that uses Δ colors if and only if there is a feasible integral schedule of size Δ. Furthermore, there is a fractional schedule of size $C = \Delta$: simply set $x_{ez} = \frac{1}{\Delta}$ for all $z = 1, \ldots, \Delta$. □

We give a better analysis of RandomSchedule covering a wider range of instances.

5.3 Improvement by the LLL

Let us study in this section the variant of resource constrained scheduling, where we omit the processor requirement and just consider the problem of finding a schedule with minimum makespan with respect to the resource constraints $R_1, \ldots, R_s$. This is a special case of them problem considered in Section 5.2. Suppose we have already applied algorithm RandomSchedule and obtained an integral solution with makespan at most $N = \lceil (1+\epsilon)C \rceil$.

Resource constrained scheduling induces the hypergraph $\mathcal{H} = (V, \mathcal{E})$ with $V = \mathcal{J}$ and $\mathcal{E} = \{E_1, \ldots, E_s\}$ such that

$$E_i = \{J_j \in \mathcal{J} \mid R_i(j) = 1\}$$

for all $i = 1, \ldots, s$.

So E_i contains all jobs which require resource R_i. Let Δ be the edge degree of $\mathcal{H}$, that is

$$\Delta = \max_{1 \le i \le s} |\{E_k \in \mathcal{E}\ ;\ E_k \cap E_i \neq \emptyset\}|.$$

The crucial point here is that for sparse hypergraphs, Δ can be much smaller than s. Define sN events by

$$\xi_{i,z} \equiv \text{``}\sum_{j=1}^{n} R_i(j) X_{jz} \ge b_i(1+\epsilon)^{-1}(1+\delta_i)\text{``} \tag{6}$$

for $1 \le i \le s$, $1 \le z \le N$ and some constant $\delta_i > 0$ to be fixed later. The dependency among these events is affected by the following two factors: each job J_j scheduled at time z

(i) will not contribute to events corresponding to times $1, \ldots, z-1, z+1, \ldots, N$ and
(ii) will contribute to at most Δ events occurring at time z.

So the dependency d is at most ΔN.

Theorem 13. *Let* $0 < \epsilon < 1$. *If* $b_i = \Omega\left(\frac{(1+\epsilon)\log(d)}{\epsilon^2}\right)$ *for all* $i = 1, \ldots, s$, *then with positive probability* RandomSchedule *generates a feasible schedule having makespan at most* $N = \lceil (1+\epsilon)C \rceil$.

Proof. After Step 2 of RandomSchedule we obtain a fractional solution satisfying $\sum_{j=1}^{n} R_i(j)\widehat{x}_{jl} \le b_i(1+\epsilon)^{-1}$ for all $i = 1, \ldots, s$ and $l = 1, \ldots, N$. Step 3 of the algorithm then proceeds with the rounding. Event $\xi_{i,z}$ occurs when after rounding, the number of units of resource R_i being used by jobs scheduled at time instance z deviates from the mean $\mu = b_i(1+\epsilon)^{-1}$ by a multiplicative factor of at least $1+\delta_i$. We invoke Theorem 4: If $b_i(1+\epsilon)^{-1} \geq \log(e\gamma(d+1))/2$ then for

$$\delta_i = \Theta\left(\sqrt{\frac{\log(e\gamma(d+1))}{b_i/(1+\epsilon)}}\right) \tag{7}$$

we get for all $1 \leq i \leq s$, $1 \leq z \leq N$

$$\mathbb{P}[\xi_{i,z}] \leq G(b_i(1+\epsilon)^{-1}, \delta_i) \leq \frac{1}{(e\gamma(d+1))} =: p \, ,$$

where $\gamma \geq 1$ is a constant. Since $ep(d+1) = \gamma^{-1} \leq 1$, we satisfy the conditions of LLL and prove that $\mathbb{P}(\bigwedge_{i=1}^{s} \bigwedge_{z=1}^{N} \xi_{i,z}^{c}] > 0$.
We now know that there exist vectors such that no event $\xi_{i,z}$ occurs. But we want to satisfy the original constraints. Therefore, we require $b_i(1+\epsilon)^{-1}(1+\delta_i) \leq b_i \; \forall i$, which is true for any $\epsilon \in (0,1)$ if

$$b_i = \Omega\left(\frac{(1+\epsilon)\log(e\gamma(d+1))}{\epsilon^2}\right) \, , \tag{8}$$

for all i.

Notice that rounding does not affect the schedule length, and for $\Delta \ll s$ we beat the lower bound on b_i in Theorem 11. □

At the moment it is an open problem whether the algorithm Random Schedule can be derandomized.

6 Inapproximability

In case $b_i = 1$ for all i, the algorithm of de la Vega and Lueker [41] gives a $(1+\varepsilon)s$ approximation. We show in the following that for $b_i = O(1)$ for all i, the approximation ratio cannot be independent of s. Let $G = (V, E)$ be a simple graph. The problem of properly coloring G with minimum colors can be viewed as an RCS problem with $|V|$ jobs and $|E|$ resources where each resource is required by exactly two jobs and exactly one unit of each resource is available. Feige and Kilian [19] showed that if $NP \not\subseteq ZPP$, then it is impossible to approximate the chromatic number of an n vertex graph within a factor of $n^{1-\varepsilon}$, for any fixed $\varepsilon > 0$, in time polynomial in n. Therefore, the same applies for the RCS problem. Since $s \leq n^2$ in simple graphs, the following holds.

Theorem 14. *For the resource constrained scheduling problem considered in this section with n jobs and s resources, there is no polynomial time approximation algorithm with approximation ratio at most $s^{\frac{1}{2}-\varepsilon}$, for any fixed $\varepsilon > 0$, unless $NP \subseteq ZPP$.*

Bibliography and Remarks: Applying Berger/Rompel's [10] extension of the method of $\log^c n$-wise independence to multi-valued random variables, a parallelization has been given in [40]. For $\tau \geq \frac{1}{\log n}$ there is an NC-algorithm which guarantees for every constant $\alpha > 1$, a $2^{\lceil \log \alpha C \rceil}/C \leq 2\alpha$-factor approximation, under the conditions $m, b_i \geq \alpha(\alpha-1)^{-1} n^{\frac{1}{2}+\tau} \lceil \log 3n(s+1) \rceil^{1/2}$ for all $i = 1, \dots, s$. Ahuja and Srivastav [1] obtained improved results with the Lovász-Local-Lemma.

For scheduling on unrelated parallel machines, results of similar flavor have been achieved by Lenstra, Shmoys and Tardos [29] and Lin and Vitter [30]. Lenstra, Shmoys and Tardos [29] gave a 2-factor approximation algorithm for the problem of scheduling independent jobs with different processing times on *unrelated* processors and also proved that there is no ρ-approximation algorithm for $\rho < 1.5$, unless $P = NP$. Lin and Vitter [30] considered the generalized assignment problem and the problem of scheduling of unrelated parallel machines. For the generalized assignment problem with resource constraint vector b they could show for every $\varepsilon > 0$ a $1 + \varepsilon$ approximation of the minimum assignment cost, which is feasible within the enlarged packing constraint $(2 + \frac{1}{\varepsilon})b$.

7 Algorithmic Versions of the LLL

7.1 The Base: 0/1 Random Variables

In the applications in Section 5 and 6 we proved approximation guarantees with the LLL. Per se it is not clear how an LLL-based existence proof can be turned into a polynomial-time algorithm. The breakthrough for this problem was made by Beck [8] in 1991.

Theorem 15. (Beck 1991) *Let $\mathcal{H} = (V, \mathcal{E})$ be a k-uniform hypergraph with $m := |\mathcal{E}|$ and edge degree at most d. Suppose that $d \leq 2^{\frac{k}{48}}$. Then a non-monochromatic 2-coloring of $\mathcal{H}$ can be constructed in $\mathcal{O}(m^{const.})$ time.*

This result is the basis of all algorithmic versions of the LLL in more general settings. We will give a sketch of the proof of Theorem 15. Before we describe Beck's algorithm let us briefly fix a class of hypergraphs for which a 2-coloring with property $\mathcal{B}$ can be constructed directly with the conditional probability method. This construction will be used as a sub-procedure in Beck's algorithm. We give a proof of Theorem 15 for sufficiently large k and m, *i.e* $k \geq 400$ and $m \geq 2^{10}$.

Theorem 16. (Probabilistic Coloring Theorem)*Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. Let $l \in \mathbb{N}$ be an integer such that $|E| \geq l$ for all $E \in \mathcal{E}$ and $|\mathcal{E}| < 2^{l-1}$. Then a non–monochromatic 2-coloring of $\mathcal{H}$ exists with positive probability.*

Proof. Let χ be a random 2-coloring of V, i.e., $\mathbb{P}[\chi(i) = 1] = \mathbb{P}[\chi(i) = 0] = \frac{1}{2}$ independently for all $i \in V$. For $E \in \mathcal{E}$ we define $\chi(E) := \sum_{i \in E} \chi(i)$. Let A_E be the event that E is monochromatic. Then,

$$\mathbb{P}[A_E] = 2 \cdot \left(\frac{1}{2}\right)^{-|E|} = 2^{1-|E|},$$

and consequently,

$$\mathbb{P}[\exists\ E \in \mathcal{E},\ A_E] \leq \sum_{E \in \mathcal{E}} \mathbb{P}[A_E] \leq |\mathcal{E}| \cdot 2^{1-|E|} < 2^{l-1} \cdot 2^{1-l} = 1.$$

Hence there exists a non-monochromatic 2-coloring. □

We wish to transform the probabilistic coloring theorem into a polynomial-time, deterministic algorithm. This algorithm should also be extended to a partial coloring procedure where we would like to have the freedom not to color some vertices.

This can be accomplished by a modification of the conditional probability method. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph and let $F_1, \ldots, F_L \subseteq V$ be some subsets. We assume that $L \leq 2^{l-1}$. Let $S \subseteq V$ be an arbitrary set which we would like to color with 2 colors and let ξ be the event that there is some monochromatic F_i with $|F_i \cap S| \geq l$. Suppose $S = \{v_1, \ldots, v_s\}$ and the vertices $v_1, \ldots, v_j$,$1 \leq j \leq s$ have colors $x_1, \ldots, x_j \in \{0,1\}$. Let $\mathbb{P}[\xi | x_1, \ldots, x_j]$ be the conditional probability that ξ occurs under the condition that $v_1, \ldots, v_j$ have been colored with colors $x_1, \ldots, x_j$.

Algorithm Partial-Color

```
Input : Hypergraph $\mathcal{H} = (V, \mathcal{E})$, $F_1, \ldots, F_L$, $S$ and $l$ as above.

For $j = 1, \ldots, n$ do
Suppose that $v_1, \ldots, v_j$ have been colored with colors $x_1, \ldots, x_j$.
  1) If $v_{j+1} \in S$, choose the color of $v_{j+1}$ to be $x_{j+1} \in \{0,1\}$ such that $x_{j+1}$ is
     the minimum of the function $\omega \to \mathbb{P}[\xi | x_1, \ldots, x_j, \omega]$.
  2) If $v_{j+1} \notin S$ and there is $F_i \in \{F_1, \ldots, F_L\}$ with $v_{j+1} \in F_i$, let $W_j := F_i \setminus S$
     and update $V$:
     a) $V := V \backslash W_j$ , $n := n - |W_j|$.
     b) Renumber the vertices such that $V = \{v_1, \ldots, v_j, v_{j+1}, \ldots, v_n\}$.
     c) Go to 1).
```

Note that without step 2, the algorithm is nothing but the basic conditional probability method which gives a non-monochromatic coloring of $F_1, \ldots, F_L$.

The reason why the incorporation of partial coloring works is simple: If we do not color some vertices, then this can never increase the conditional probability of producing a monochromatic hyperedge. In other words, the conditional probability $\mathbb{P}[\xi | x_1, \ldots, x_j]$ under a partial coloring is at most $\mathbb{P}[\xi | x_1, \ldots, x_j]$ under a full coloring.

Theorem 17. (Basic Coloring Theorem)*Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph and $F_1, \ldots, F_L$ be subsets of V. Let $l \in \mathbb{N}$ with $L \leq 2^{l-1}$. Then* Partial-Color *builds in polynomial-time a 2-colored subset $S \subseteq V$ such that no F_i with $|F_i \cap S| \geq l$ is monochromatic.*

The conditional probabilities can be computed, but for the sake of simplicity, we omit this computation.. We only note that the conditional probability $\mathbb{P}[\xi | x_1, \ldots, x_j]$

(a) depends on l, L and $x_1, \ldots, x_j$,
(b) does not depend on $v_{j+1}, \ldots, v_n$.

The deterministic algorithm of Beck starts with a partial coloring and iterates until a complete coloring is obtained. A partial coloring of $\mathcal{H} = (V, \mathcal{E})$ is a mapping $\chi : V \to \{-1, 0, 1\}$, and we identify $1, -1$ with colors, say red and blue, and 0 with a non-color. Let a partial coloring be given and let $\gamma \geq 2$. An edge $E \in \mathcal{E}$ is called *dangerous*, with respect to this partial coloring if it has $\lceil k/\gamma \rceil$ points in one color class. $\lceil k/\gamma \rceil$ is called a threshold value.

In the following we describe Beck's algorithm along with its analysis, leading to a proof of Theorem 7.1. For simplicity we assume k to be divisible by $2, 4$ and 6 and consider in Theorem 7.1 the stronger degree condition $d \leq 2^{\frac{k}{96}}$.

Algorithm $\mathcal{H}-$Color

1. *First Pass.* Threshold value is $k/2$.

a) *Partial Coloring.* Choose L, l and a set $\mathcal{F}$ of subsets $F_1, \ldots, F_L \subseteq V$ for all $i = 1, \ldots, L$. This gives a hypergraph $\mathcal{G} = (V, \mathcal{F})$. We run Partial-Color on the hypergraph $\mathcal{G}$ with parameters l and L, and sequentially color the points of V in the following way: If we reach an uncolored point $v \in V$, such that some $E \in \mathcal{E}$ containing v is dangerous, then we remove v and all uncolored vertices in dangerous hyperedges of $\mathcal{H} = (V, \mathcal{E})$, and proceed to the next uncolored vertex. Let $S \subseteq V$ be the set of the colored points at the end of the first pass.

b) *Truncation.* All bicolored edges of $\mathcal{H}$ will never become monochromatic, and we may remove them from $\mathcal{E}$. Furthermore, we remove the colored points from V. Let $\mathcal{H}^{(1)} = (V^{(1)}, \mathcal{E}^{(1)})$ be the so obtained new hypergraph (where $V^{(1)} = V \backslash S$, $\mathcal{E}^{(1)} = (\mathcal{E} \backslash \{\text{bichromatic edges}\})|_{V^{(1)}}$). Observe that every edge in $\mathcal{H}^{(1)}$ has at least $k/2$ points.

Before we proceed with the algorithm let us examine $\mathcal{H}^{(1)}$ more closely. We expect that $\mathcal{H}^{(1)}$ is sparser than $\mathcal{H}$ in the sense that it has fewer vertices and edges. But not only that. $\mathcal{H}^{(1)}$ has a less congested dependency structure compared with $\mathcal{H}$. The whole analysis is based on the following fundamental observation about the structure of the truncated hypergraph.

Lemma 3 (Main Lemma). *Every connected component of $\mathcal{D}(\mathcal{H}^{(1)})$ has size at most $\frac{\beta \log m}{k} \cdot 2^{4\alpha k}$, where α, β are constants chosen such that*

$$1 + 6\alpha\beta + 8\beta/k \leq \beta/2.$$

Proof (Sketch). Let $\mathcal{D}(\mathcal{H}^{(1)})$ resp. $\mathcal{D}(\mathcal{H})$ be the dependency graph of $\mathcal{H}^{(1)}$ resp. $\mathcal{H}$ (see Definition 2). For any two vertices i and j, $i \neq j$, in the vertex set

$V(\mathcal{D}(\mathcal{H}))$ of $\mathcal{D}(\mathcal{H})$ let $\delta(i,j)$ be the length of the shortest path between them. Let $\mathcal{D}(\mathcal{H})^{(a,b)} = (\mathcal{V}, \mathcal{E})$ be a graph with $\mathcal{V} = V(\mathcal{D}(\mathcal{H}))$ and $E = \{(i,j) \mid i,j \in \mathcal{V},\ i \neq j$ and $a \leq \delta(i,j) \leq b\}$. We call $T \subseteq \mathcal{V}$ an (a,b)-tree if the subgraph induced by T in $\mathcal{D}(\mathcal{H})^{(a,b)}$ is connected.

Let L be the number of $(2,6)$-trees of size $\beta \log m/k$. Every $(2,6)$-tree forms a set $F \subseteq V$. Let $\mathcal{F} = \{F_1, \ldots, F_L\}$ be the set of these subsets of V. Let $l = 1 + \beta \log m/2$ and consider the hypergraph $\mathcal{G} = (V, \mathcal{F})$. The partial coloring algorithm is applied to $\mathcal{G}$ with parameters L and l to build a colored set $S \subseteq V$. By the basic coloring theorem (Theorem 17) every $F \cap S$, $F \in \mathcal{F}$ is non-monochromatic, provided that $|F \cap S| \geq l$ and $L \leq 2^{l-1}$. In order to satisfy this condition we bound L. The number of $(2,6)$-trees of size $\beta \log m/k$ in a graph with m vertices is at most $2^{(1+6\alpha\beta+8\frac{\beta}{k})\log m}$ [8]. Now, if we choose β in a way that

$$L \leq 2^{(1+6\alpha\beta+8\frac{\beta}{k})\log m} \leq 2^{\frac{\beta \log m}{2}} = 2^{l-1}, \tag{9}$$

i.e $1 + 6\alpha\beta + 8\frac{\beta}{k} \leq \beta \log m/2$, then the conditions of the basic coloring theorem are satisfied.

To finish the proof, let us assume for a moment that there is a connected component of $\mathcal{D}(\mathcal{H}^{(1)})$ of size at least $2^{4\alpha k}(\beta \log m/k)$, where $\alpha = 1/48$. It can be shown that there exists a $(2,6)$-tree of size $\beta \log m/k$ such that its corresponding set $F^* \in \mathcal{F}$ satisfies $|F^* \cap S| \geq l$, but F^* is monochromatic contradicting the just proved fact that such sets are non-monochromatic. □

Now we may continue the coloring of $\mathcal{H}^{(1)}$.

c) Colorability Test. By the main lemma, taking $\alpha = 1/48$ and $\beta = 4$, $\mathcal{D}(\mathcal{H}^{(1)})$ (and therefore $\mathcal{H}^{(1)}$) breaks into components, say $C_1, \ldots, C_r$ of size at most

$$f_1 := \frac{4 \log m}{k} \cdot 2^{\frac{k}{12}}.$$

Let $C \in \{C_1, \ldots, C_r\}$ and let $\mathcal{H}_C^{(1)} = (V_{1,C}, \mathcal{E}_C^{(1)})$ be the subhypergraph of $\mathcal{H}^{(1)}$ having only the hyperedges from C.

Case 1: $f_1 \leq 2^{\frac{k}{4}}$. Then $|\mathcal{E}_C^{(1)}| \leq f_1 \leq 2^{\frac{k}{4}}$, and since every hyperedge from $\mathcal{E}_C^{(1)}$ has at least $k/2$ vertices, by Theorem 17 we can find a non-monochromatic coloring for all $\mathcal{H}_C^{(1)}$, $C \in \{C_1, \ldots, C_r\}$
Case 2: $f_1 > 2^{\frac{k}{4}}$. In this case the size of the connected components of $\mathcal{D}(\mathcal{H}^{(1)})$ is even smaller: $\frac{4 \log m}{k} \cdot 2^{\frac{k}{12}} > 2^{\frac{k}{4}}$ implies $\frac{4 \log m}{k} > 2^{\frac{k}{6}}$. Hence

$$|\mathcal{E}_C^{(1)}| < \frac{4 \log m}{k} \left(\frac{4 \log m}{k}\right)^{\frac{1}{2}} = \left(\frac{4 \log m}{k}\right)^{\frac{3}{2}} \leq (\log m)^{\frac{3}{2}}$$

for $k \geq 4$. We enter the second pass of the algorithm.

2. Second Pass. Threshold value is $\frac{k}{4}$. With $\mathcal{H}^{(1)}$ we go through steps $a)$ and $b)$ of the first pass. Let $\mathcal{H}^{(2)} = (V^{(2)}, \mathcal{E}^{(2)})$ be the hypergraph after the truncation step.

Let us consider $\mathcal{D}(\mathcal{H}^{(2)})$. We apply the main lemma to $\mathcal{D}(\mathcal{H}^{(2)})$ choosing $\alpha = \frac{1}{96}$ and $\beta = 6$: Every connected component of $\mathcal{D}(\mathcal{H}^{(2)})$ has size at most

$$\frac{6\log[(\log m)^{\frac{3}{2}}]}{k/2} \cdot 2^{\frac{k}{24}} \le \underbrace{\frac{18 \log\log m}{k} \cdot 2^{\frac{k}{24}}}_{:=f_2} .$$

d) *Colorability Test.* Each hyperedge of $\mathcal{D}(\mathcal{H}^{(2)})$ has size at least $k/4$.

Case 1: If $f_2 < 2^{\frac{k}{12}} = 2^{\frac{k/6}{2}}$, we color all points of $\mathcal{H}^{(2)}$ with Theorem 17.
Case 2: If $f_2 \ge 2^{\frac{k}{12}}$, then $\frac{18\log\log m}{k} \ge 2^{\frac{k}{24}}$, so the size of the components of $\mathcal{D}(\mathcal{H}^{(2)})$ is at most

$$\frac{18\log\log m}{k} \cdot 2^{\frac{k}{24}} \le \left(\frac{18\log\log m}{k}\right)^2 \le \frac{\log m}{k}, \tag{10}$$

since $k \ge 400$ and $\log m \ge 2^{10}$.

e) Brute-Force Coloring.

Let C be a component of $\mathcal{D}(\mathcal{H}^{(2)})$, let $\mathcal{E}_C^{(2)}$ be the set of all hyperedges corresponding to C and let $V_C^{(2)} \subseteq V^{(2)}$ be the set of all points from the hyperedges of $\mathcal{E}_C^{(2)}$. By (10) we have

$$|V_C^{(2)}| \le k \cdot \frac{\log m}{k} = \log m. \tag{11}$$

Hence, the number of 2-colorings of $\mathcal{H}_C^{(2)}$ is $2^{\mathcal{O}(\log m)} = m^{\mathcal{O}(1)}$. Now we can test in polynomial-time $m^{\mathcal{O}(1)}$ whether or not there is a non-monochromatic coloring among them. But is there such a coloring? Fortunately, the Local-Lemma proves the existence of at least one non-monochromatic coloring, because

$$\deg(\mathcal{H}^{(2)}) \le \deg(\mathcal{H}) \le 2^{\frac{k}{96}} \le 2^{\frac{k}{4}-3}.$$

This finishes the proof of Theorem 7.1. □

7.2 Extension I: General Random Variables – Small Domains

Molloy and Reed [31] generalized Beck's algorithm and gave new applications. Their main result can be formulated as follows.

Let $f_1, \dots, f_m$ be independent random variables where each f_i takes values in some domain of cardinality at most γ. Suppose furthermore that we are given n "bad" events $A_1, \dots, A_n$ where A_i is determined by the random variables in some set $F_i \subseteq \{f_1, \dots, f_m\}$, and let $w \ge \max_{i=1,\dots,m} |F_i|$. We say, A_1 depends on A_j iff $F_i \cap F_j \ne \emptyset$. Let d_i be the number of A_j's on which A_i depends and let

$$d = \max_{1\le i\le n} d_i \quad \text{and} \quad p = \max_{1\le i \le n} \mathbb{P}[A_i].$$

Some assumption on the computation time of the probabilities is necessary. Let t_1 be the time to carry out the random trial f_i, $1 \leq i \leq m$ and let t_2 the time to compute the conditional probabilities $\mathbb{P}[A_i | f_{j_1} = w_1, \ldots, f_{j_k} = w_k]$ for $f_{j_1}, \ldots, f_{j_k} \in F_i$ and $w_1, \ldots, w_k$ in the domains of the $f_{j_1}, \ldots, f_{j_k}$.

Theorem 18. (Molloy, Reed 1998) *If* $pd^9 < 1/8$*, then an assignment of all the* f_i *can be found with a randomized* $\mathcal{O}(md(t_1+t_2)+m\gamma^{wd \log\log m})$*−time algorithm such that* $\mathbb{P}\left[\bigcap_{i=1}^{n} A_i^c\right] > 0$.

Roughly speaking the result says that whenever the stronger condition $8pd^9 < 1$ instead of the Local-Lemma condition $4pd \leq 1$ holds, at least a randomized algorithmic version of the Local Lemma in a more general framework can be given. The proof is based on a variant of Beck's algorithm.

Among the various striking applications of the Local Lemma is certainly acyclic edge coloring. A proper edge coloring of a graph is *acyclic* if the union of two color classes is a forest in the graph. Alon [3] proved:

Theorem 19. (Alon 1991) *If* G *is a graph with maximum vertex degree* Δ*, then* G *has an acyclic edge coloring using at most* 16Δ *colors.*

Molloy and Reed showed with their method how this result can be made constructive.

7.3 Extension II: General Random Variables – Large Domains

The algorithm of Molloy and Reed does not run in polynomial-time, if γ, the size of the domains of the random variables is large, say $\gamma = (m+n)^c$, $c > 0$ a constant. Here Leighton, Lu, Rao and Srinivasan [28] gave constructive and efficient extensions and covered new applications like the disjoint path problem in expander graphs, hypergraphs partitioning and routing with low congestion. The last two problems can be formulated as integer linear programs, sometimes called Minmax Integer Program(MIP) in literature.

Given positive integers k and l_i , $i = 1, \ldots, k$, let $N = \sum_{i=1}^{k} l_i$. An MIP aims to minimize a variable $W \in \mathbb{R}$ such that

(i) $Ax \leq \vec{W}$, where $A \in [0,1]^{m \times N}$, x is an N-dimensional vector consisting of variables $x_{i,j}$, $i = 1, \ldots, k$, $j = 1, \ldots, l_i$ and $\vec{W}$ is an m-dimensional vector having W as each of its components,
(ii) $\sum_{j=1}^{l_i} x_{i,j} = 1 \; \forall i = 1, \ldots, k$ and
(iii) $x_{i,j} \in \{0,1\} \; \forall i, j$.

MIPs are NP-hard, so our aim is to relax the integrality constraints to $x_{i,j} \in [0,1] \; \forall i, j$ and then solve the LP-relaxation in polynomial-time. Let $W^* \in \mathbb{R}$ be the optimum value of the objective function of our LP-relaxation, $\vec{W}^*$ the m-dimensional vector having W^* as its components and let $x^* \in [0,1]^N$ be an optimal solution. The idea now is to round the fractional solution to obtain an integral solution and verify the quality of this integral solution. For each $i = 1, \ldots, k$ randomly and independently round exactly one $x^*_{i,1}, \ldots, x^*_{i,l_i}$ out of the l_i variables to 1, *i.e.,* for each i, j we have $\mathbb{P}[x_{i,j} = 1] = x^*_{i,j}$. If x_{i_0,j_0} is rounded to 1, then because of constraint (ii) of the integer program $x_{i_0,j} = 0$, $\forall j \in \{1, \ldots, l_{i_0}\} - \{j_0\}$.

There are four important parameters which have to be taken into consideration :

(a) d, the maximum number of nonzero entries in any column of matrix A,
(b) $\tau \geq 1$, the inverse of the minimum nonzero entry of A,
(c) $l = \max_{i \in \{1,\ldots,k\}} |\{x^*_{i,j} \mid 0 < x^*_{i,j} < 1\ ,\ j = 1, \ldots, l_i\}|$ and
(d) t, the maximum number of variables to be rounded in a row of the constraints $Ax \leq \vec{W}^*$.

Notice that a row $r \in \{1, \ldots, m\}$ of the constraints in (i) depends on another row s if there exist $i \in \{1, \ldots, k\}$ and $j_1, j_2 \in \{1, \ldots, l_i\}$ such that $A_{r,(i,j_1)}, A_{s,(i,j_2)} \neq 0$ and $x^*_{i,j_1}\ ,\ x^*_{i,j_2} \notin \{0, 1\}$. So each row can be affected by at most dlt other rows. Let $\alpha = H(W^*, 1/dlt)$ (recall Theorem 4) and let $\xi_1, \ldots, \xi_m$ be the events "$(Ax)_i > W^*(1 + c\alpha)$", $i = 1, \ldots, m$, and let $c > 0$ be a constant. With the LLL we can show that randomized rounding works:

Proposition 1. *For a sufficiently large constant $c > 0$, we have*

$$\mathbb{P}[\bigcap_{r=1}^{m} \bar{\xi}_r] > 0,$$

thus the vector $x \in \{0,1\}^N$ generated by randomized rounding satisfies the MIP with constraint (i) replaced by the weaker inequality $Ax \leq \vec{W}^*(1 + c\alpha)$.

Observe that $H(x, y) \geq H(x, z)$ if $y \geq z$. Leighton et al. [28] succeed in decreasing l and t to $poly(W^*, d, \tau)$ at the cost of a marginal increase in W^* by performing rounding in several iterations leading to the improved result:

Theorem 20. *There exists a deterministic polynomial-time algorithm for finding a solution $x \in \{0,1\}^N$ with $Ax \leq \vec{W}^*(1 + c'\alpha') + O(1)$, where*

$$\alpha' = H\left(W^*, \frac{1}{poly(W^*, d, \tau)}\right) \leq \alpha\ ,$$

and $c' > 0$ is a constant.

Observe that the parameter γ (the cardinality of the domain of independent random variables) in the approach of Molloy and Reed can be m, thus the running time there can be subexponential for this problem.

7.4 Resource Constrained Scheduling Revisited

For RCS we cannot use Theorem 20 directly as we have to deal with packing constraints of the type "$Ax \leq b$" with a fixed vector b. Nevertheless, a similar approach [1], can be used to prove the following derandomized counterpart of Theorem 13.

Theorem 21. *Let a resource constrained scheduling problem as in Section 5.3 be given. For any $\epsilon \in (0,1)$, a feasible schedule with makespan at most $N = \lceil (1+\epsilon)C \rceil$ can be found in polynomial-time provided that $b_i = \Omega\left(\frac{(1+\epsilon)\log(dN)}{\epsilon^2}\right)$ for all $i = 1, \dots, s$.*

On the one hand we have the $(1+\varepsilon)s$ approximation of de la Vega and Lueker [41] when $b_i = 1$ for all i, and on the other hand we have our $(1+\varepsilon)$ approximation for $b_i = \Omega((1+\varepsilon)\varepsilon^{-2}\log d)$ (Theorem 13). However, we do not know the approximation quality for all values of b_i. The most challenging open question here seems to be the following: how does the approximation ratio behave when $b_i \in (1, \log d]$ for all i?

References

1. N. Ahuja, A. Srivastav; *On Constrained hypergraph coloring and scheduling.* Proceedings of the 5th International Workshop on Approximation Algorithms and Combinatorial Optimization, Rome (2002), Springer Lecture Notes in Computer Science, Vol. 2462, 14–25.
2. N. Alon. *The linear arboricity of graphs.* Israel J. Math. **62** (1988), 311–325.
3. N. Alon. *A parallel algorithmic version of the Local Lemma.* Random Structures and Algorithms **2**:4 (1991), 367–378.
4. N. Alon, C. McDiarmid, B. Reed. *Star arboricity.* Combinatorica **12**:4 (1992), 375-380.
5. N. Alon, C. McDiarmid, B. Reed. *Acyclic coloring of graphs.* Random Structures and Algorithms **2**:3 (1991), 277–288.
6. N. Alon, J. Spencer, P. Erdős. *The probabilistic method.* John Wiley & Sons, Inc. 1992.
7. D. Angluin, L.G. Valiant. *Fast probabilistic algorithms for Hamiltonian circuits and matchings.* J. Computer System Sciences **18** (1979), 155–193.
8. J. Beck. *An algorithmic approach to the Lovász Local Lemma I.* Random Structures & Algorithms **2**:4 (1991), 343–365.
9. C. Berge. *Graphs and Hypergraphs.* North Holland, Amsterdam 1973.
10. B. Berger, J. Rompel. *Simulating ($\log^c n$)-wise independence in NC.* J.ACM **38**:4 (1991), 1026–1046. Preliminary Version in: Proceedings of the IEEE Symposium on the Foundation of Computer Science 1989, (FOCS '89).
11. C. Bertram-Kretzberg, H. Lefmann. *The algorithmic aspects of uncrowded hypergraphs.* SIAM J. Computing **29**:1 (1998), 201–230.
12. B. Bollobás. *Modern Graph Theory.* Springer, New York 1998.
13. A. Z. Broder, A. M. Frieze, E. Upfal. *Existence and construction of edge disjoint paths on expander graphs.* SIAM J. Computing **23**:5 (1994), 976-989. Preliminary Version in: Proceedings of the 24th Annual ACM Symposium on the Theory of Computing 1992, (STOC '92), 140–149.

14. H. Chernoff. *A measure of asymptotic efficiency for test of a hypothesis based on the sum of observation.* Ann. Math. Stat. **23** (1952), 493–509.
15. A. Czumaj, C. Scheideler. *Coloring non-uniform hypergraphs: a new algorithmic approach to the general Lovász Local Lemma.* Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms 2000, (SODA 2000), 30 – 39.
16. A. Czumaj, C. Scheideler. *An algorithmic approach to the general Lovász Local Lemma with applications to scheduling and satisfiability problems* Proceedings of the 32nd ACM Symposium on Theory of Computing 2000, (STOC 2000), 38 – 47.
17. R. Diestel. *Graph Theory.* Springer, New York 1997.
18. P. Erdős, L. Lovász. *Problems and results on 3 - chromatic hypergraphs and some related questions.* Infinite and finite sets, A. Hajnal et al., Colloq. Math. Soc. J. Bolyai, Vol. II. (1975), 609–627.
19. U. Feige, J. Kilian. *Zero knowledge and the chromatic number.* Journal of Computer and System Sciences **57** (1998), 187–199.
20. A. D. Fundia. *Derandomizing Chebyshev's Inequality to find Independent Sets in Uncrowded Hypergraphs.* Random Structures & Algorithms **8** (1996), 131–147.
21. M. R. Garey, R. L. Graham, D. S. Johnson, A.C.-C. Yao. *Resource constrained scheduling as generalized bin packing.* J. Combinatorial Theory Ser. A **21** (1976), 257–298.
22. L. M. Goldberg, M. Paterson, A. Srinivasan, E. Sweedyk. *Better approximation guarantees for job-shop scheduling.* SIAM Journal on Discrete Mathematics **14**:1 (2001), 67–92. Preliminary Version in: Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms 1997, (SODA'97), 599–608.
23. M. Habib, C. McDiarmid, J. Ramirez-Alfonsin and B. Reed. *Probabilistic methods for algorithmic discrete mathematics.* Springer Series *Algorithms and Combinatorics, Vol. 16*, Springer Verlag, 1998.
24. L.A. Hall. *Approximation algorithms for scheduling.* Approximation algorithms for NP-hard problems, D.S.Hochbaum(ed.), PWS Publishing Company, Boston 1997, 1–46.
25. W. Hoeffding. *Probability inequalities for sums of bounded random variables.* J. Amer. Statist. Assoc. **58** (1963), 13–30.
26. S. Janson, T. Łuczak, A. Ruciński. *Random Graphs.* Wiley-Interscience Series in Discrete Mathematics and Otimization, John Wiley & Sons, Inc. New York Toronto 2000.
27. T. Leighton, B. Maggs, S. Rao. *Packet routing and job-shop scheduling in O(Congestion+Dilation) steps.* Combinatorica **14** (1994), 167–186.
28. T. Leighton, Chi-Jen Lu, S. Rao, A. Srinivasan. *New algorithmic aspects of the local lemma with applications to routing and partitioning.* SIAM Journal on Computing **31** (2001), 626–641.
29. J. K. Lenstra, D. B. Shmoys, E. Tardos. *Approximating algorithms for scheduling unrelated parallel machines.* Math. Programming **46** (1990), 259–271.
30. J.-H. Lin, J. S. Vitter. *ε-approximations with minimum packing constraint violation.* Proceedings 24th Annual ACM Symposium on the Theory of Computation 1992, (STOC '92), 771–782.
31. M. Molloy, B. Reed. *Further algorithmic aspects of the Local Lemma.* in: Proceedings of the 30th ACM-Symposium on the Theory of Computing 1998, (STOC '98), 524 – 530.
32. M. Okamoto. *Some inequalities relating to the partial sum of binomial probabilities.* Ann. Inst. Statist. Math. **10** (1958), 29–35.
33. P. Raghavan, C.D. Thompson. *Randomized Rounding.* Combinatorica **7** (1987), 365–374.

34. H. Röck, G. Schmidt. *Machine aggregation heuristics in shop scheduling.* Methods Oper. Res. **45** (1983), 303–314.
35. D.B. Shmoys, C. Stein, J. Wein. *Improved approximation algorithms for shop scheduling problems.* SIAM J. Computing **23** (1994), 617–632.
36. A. Srinivasan. *An extension of the Lovász Local Lemma, and its application to integer programming.* Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms 1996, (SODA '96), 6–15.
37. A. Srinivasan. *Improved approximation guarantees for packing and covering integer programs.* SIAM J. Computing **29** (1999), 648–670.
38. A. Srivastav. *Derandomization in Combinatorial Optimization.* In: Handbook of Randomized Computing Volume II. Series: Combinatorial Optimization. Vol. **9**. S. Rajasekaran, P. M. Pardalos, J. H. Reif, J. D. Roli (Eds.) (2001).
39. A. Srivastav, P. Stangier. *Algorithmic Chernoff-Hoeffding inequalties in integer programming.* Random Structures & Algorithms **8**:1 (1996), 27–58.
40. A. Srivastav, P. Stangier. *Tight aproximations for resource constrained scheduling and bin packing.* Discrete Appl. Math. **79** (1997), 223–245.
41. W. F. de la Vega, C. S. Luecker. *Bin packing can be solved within* $1 - \varepsilon$ *in linear time.* Combinatorica **1** (1981), 349–355.
42. D. B. West. *Introduction to Graph Theory.* Prentice-Hall, Inc. Simon & Schuster 1996.

Author Index