Philip S. Yu · Jiawei Han
Christos Faloutsos *Editors*

Link Mining: Models, Algorithms, and Applications

Springer

Link Mining: Models, Algorithms, and Applications

Philip S. Yu · Jiawei Han · Christos Faloutsos
Editors

Link Mining: Models, Algorithms, and Applications

Springer

Editors
Philip S. Yu
Department of Computer Science
University of Illinois at Chicago
851 S. Morgan St.
Chicago, IL 60607-7053, USA
psyu@cs.uic.edu

Jiawei Han
Department of Computer Science
University of Illinois at
Urbana-Champaign
201 N. Goodwin Ave.
Urbana, IL 61801, USA
hanj@cs.uiuc.edu

Christos Faloutsos
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213, USA
christos@cs.cmu.edu

ISBN 978-1-4419-6514-1 e-ISBN 978-1-4419-6515-8
DOI 10.1007/978-1-4419-6515-8
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010932880

Preface

With the recent flourishing research activities on Web search and mining, social network analysis, information network analysis, information retrieval, link analysis, and structural data mining, research on link mining has been rapidly growing, forming a new field of data mining.

Traditional data mining focuses on "flat" or "isolated" data in which each data object is represented as an independent attribute vector. However, many real-world data sets are inter-connected, much richer in structure, involving objects of heterogeneous types and complex links. Hence, the study of link mining will have a high impact on various important applications such as Web and text mining, social network analysis, collaborative filtering, and bioinformatics.

As an emerging research field, there are currently no books focusing on the theory and techniques as well as the related applications for link mining, especially from an interdisciplinary point of view. On the other hand, due to the high popularity of linkage data, extensive applications ranging from governmental organizations to commercial businesses to people's daily life call for exploring the techniques of mining linkage data. Therefore, researchers and practitioners need a comprehensive book to systematically study, further develop, and apply the link mining techniques to these applications.

This book contains contributed chapters from a variety of prominent researchers in the field. While the chapters are written by different researchers, the topics and content are organized in such a way as to present the most important models, algorithms, and applications on link mining in a structured and concise way. Given the lack of structurally organized information on the topic of link mining, the book will provide insights which are not easily accessible otherwise. We hope that the book will provide a useful reference to not only researchers, professors, and advanced level students in computer science but also practitioners in industry.

We would like to convey our appreciation to all authors for their valuable contributions. We would also like to acknowledge that this work is supported by NSF through grants IIS-0905215, IIS-0914934, and DBI-0960443.

Chicago, Illinois — Philip S. Yu
Urbana-Champaign, Illinois — Jiawei Han
Pittsburgh, Pennsylvania — Christos Faloutsos

Contents

Contributors

Tanya Berger-Wolf University of Illinois at Chicago, Chicago, IL 60607, USA

Kuiyu Chang School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore

Chen Chen University of Illinois at Urbana-Champaign, Urbana, IL, USA

Hong Cheng The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

Young-Rae Cho Baylor University, Waco, TX 76798, USA

Mihai Christodorescu IBM T. J. Watson Research Center, Hawthorne, NY, USA

Kenneth L. Clarkson IBM Almaden Research Center, San Jose, CA, USA

J. Perren Cobb Department of Anesthesia, Critical Care, and Pain Medicine, Massachusetts General Hospital, Boston, MA 02114, USA

Anwitaman Datta School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore

Pedro Domingos Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350, USA

Christos Faloutsos Carnegie Mellon University, Pittsburgh, PA 15213, USA

Matt Fredrikson University of Wisconsin at Madison, Madison, WI, USA

Lise Getoor Department of Computer Science, University of Maryland, College Park, MD, USA

Zhen Guo Computer Science Department, SUNY Binghamton, Binghamton, NY, USA

Jiawei Han UIUC, Urbana, IL, USA

David Kempe University of Southern California, Los Angeles, CA 90089, USA

Stanley Kok Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350, USA

Ravi Kumar Yahoo! Research, 701 First Ave, Sunnyvale, CA 94089, USA

Xin Li Microsoft Corporation One Microsoft Way, Redmond, WA 98052, USA

Ee-Peng Lim School of Information Systems, Singapore Management University, Singapore

Cindy Xide Lin University of Illinois at Urbana-Champaign, Urbana, IL, USA

Bing Liu Department of Computer Science, University of Illinois at Chicago, 851 S. Morgan (M/C 152), Chicago, IL 60607-7053, USA

Huan Liu Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-8809, USA

Kun Liu Yahoo! Labs, Santa Clara, CA 95054, USA

Bo Long Yahoo! Labs, Yahoo! Inc., Sunnyvale, CA, USA

Daniel Lowd Department of Computer and Information Science, University of Oregon, Eugene, OR 97403-1202, USA

Galileo Mark Namata Department of Computer Science, University of Maryland, College Park, MD, USA

Aniruddh Nath Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350, USA

Arye Nehorai Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130, USA

Jasmine Novak Yahoo! Research, 701 First Ave, Sunnyvale, CA 94089, USA

Feng Pan Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA

Spiros Papadimitriou IBM TJ. Watson, Hawthorne, NY, USA

Jignesh M. Patel University of Wisconsin, Madison, WI 53706-1685, USA

Hoifung Poon Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350, USA

Raghu Ramakrishnan Yahoo! Research, Santa Clara, CA, USA

Matthew Richardson Microsoft Research, Redmond, WA 98052, USA

Pinaki Sarder Department of Computer Science and Engineering, Washington University in St.Louis, St. Louis, MO 63130, USA

Hossam Sharara Department of Computer Science, University of Maryland, College Park, MD, USA

Parag Singla Department of Computer Science, The University of Texas at Austin, 1616 Guadalupe, Suite 2408, Austin, TX 78701-0233, USA

Aixin Sun School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore

Jimeng Sun IBM TJ Watson Research Center, Hawthorne, NY, USA

Yizhou Sun University of Illinois at Urbana-Champaign, Urbana, IL, USA

Lei Tang Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-8809, USA

Chayant Tantipathananandh University of Illinois at Chicago, Chicago, IL 60607, USA

Evimaria Terzi Computer Science Department, Boston University, Boston, MA, USA

Yuanyuan Tian IBM Almaden Research Center, San Jose, CA, USA

Andrew Tomkins Google, Inc., 1600 Amphitheater Parkway, Mountain View, CA 94043, USA

Hanghang Tong Carnegie Mellon University, Pittsburgh, PA 15213, USA

Wei Wang Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA

Tianbing Xu Computer Science Department, SUNY Binghamton, Binghamton, NY, USA

Xifeng Yan University of California at Santa Barbara, Santa Barbara, CA, USA

Xiaoxin Yin Microsoft Research, Redmond, WA 98052, USA

Philip S. Yu Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA

Aidong Zhang State University of New York at Buffalo, Buffalo, NY 14260, USA

Weixiong Zhang Departments of Computer Science and Engineering and Genetics, Washington University in St. Louis, St. Louis, MO 63130, USA

Xiang Zhang Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA

Zhongfei (Mark) Zhang Computer Science Department, SUNY Binghamton, Binghamton, NY, USA

Feida Zhu University of Illinois at Urbana-Champaign, Urbana, IL, USA

Part I
Link-Based Clustering

Chapter 1
Machine Learning Approaches to Link-Based Clustering

Zhongfei (Mark) Zhang, Bo Long, Zhen Guo, Tianbing Xu, and Philip S. Yu

Abstract We have reviewed several state-of-the-art machine learning approaches to different types of link-based clustering in this chapter. Specifically, we have presented the spectral clustering for heterogeneous relational data, the symmetric convex coding for homogeneous relational data, the citation model for clustering the special but popular homogeneous relational data—the textual documents with citations, the probabilistic clustering framework on mixed membership for general relational data, and the statistical graphical model for dynamic relational clustering. We have demonstrated the effectiveness of these machine learning approaches through empirical evaluations.

1.1 Introduction

Link information plays an important role in discovering knowledge from data. For link-based clustering, machine learning approaches provide pivotal strengths to develop effective solutions. In this chapter, we review several specific machine learning techniques to link-based clustering in two specific paradigms—the deterministic approaches and generative approaches. We by no means mean that these techniques are exhaustive. Instead, our intention is to use these exemplar approaches to showcase the power of machine learning techniques to solve different link-based clustering problems.

When we say link-based clustering, we mean the clustering of relational data. In other words, links are the relations among the data items or objects. Consequently, in the rest of this chapter, we use the terminologies of link-based clustering and relational clustering exchangeably. In general, relational data are those that have link information among the data items in addition to the classic attribute information for the data items. For relational data, we may categorize them in terms of the type of their relations [37] into homogeneous relational data (relations exist among the same type of objects for all the data), heterogeneous relational data (relations only

Z. Zhang (✉)
Computer Science Department, SUNY, Binghamton, NY, USA
e-mail: zhongfei@cs.binghamton.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_1,

exist between data items of different types), general relational data (relations exist both among data items of the same type and between data items of different types), and dynamic relational data (there are time stamps for all the data items with relations to differentiate from all the previous types of relational data which are static). For all the specific machine learning approaches reviewed in this chapter, they are based on the mathematical foundations of matrix decomposition, optimization, and probability and statistics theory.

In this chapter, we review five specific different machine learning techniques tailored for different types of link-based clustering. Consequently, this chapter is organized as follows. In Section 1.2 we study the deterministic paradigm of machine learning approaches to link-based clustering and specifically address solutions to the heterogeneous data clustering problem and the homogeneous data clustering problem. In Section 1.3, we study the generative paradigm of machine learning approaches to link-based clustering and specifically address solutions to a special but very popular problem of the homogeneous relational data clustering, i.e., the data are the textual documents and the link information is the citation information, the general relational data clustering problem, and the dynamic relational data clustering problem. Finally, we conclude this chapter in Section 1.4.

1.2 Deterministic Approaches to Link-Based Clustering

In this section, we study deterministic approaches to link-based clustering. Specifically, we present solutions to the clustering of the two special cases of the two types of links, respectively, the heterogeneous relational clustering through spectral analysis and homogeneous relational clustering through convex coding.

1.2.1 Heterogeneous Relational Clustering Through Spectral Analysis

Many real-world clustering problems involve data objects of multiple types that are related to each other, such as Web pages, search queries, and Web users in a Web search system, and papers, key words, authors, and conferences in a scientific publication domain. In such scenarios, using traditional methods to cluster each type of objects independently may not work well due to the following reasons.

First, to make use of relation information under the traditional clustering framework, the relation information needs to be transformed into features. In general, this transformation causes information loss and/or very high dimensional and sparse data. For example, if we represent the relations between Web pages and Web users as well as search queries as the features for the Web pages, this leads to a huge number of features with sparse values for each Web page. Second, traditional clustering approaches are unable to tackle with the interactions among the hidden structures of different types of objects, since they cluster data of single type based on static

features. Note that the interactions could pass along the relations, i.e., there exists influence propagation in multi-type relational data. Third, in some machine learning applications, users are not only interested in the hidden structure for each type of objects but also the global structure involving multi-types of objects. For example, in document clustering, except for document clusters and word clusters, the relationship between document clusters and word clusters is also useful information. It is difficult to discover such global structures by clustering each type of objects individually.

Therefore, heterogeneous relational data have presented a great challenge for traditional clustering approaches. In this study [36], we present a general model, the collective factorization on related matrices, to discover the hidden structures of objects of different types based on both feature information and relation information. By clustering the objects of different types simultaneously, the model performs adaptive dimensionality reduction for each type of data. Through the related factorizations which share factors, the hidden structures of objects of different types may interact under the model. In addition to the cluster structures for each type of data, the model also provides information about the relation between clusters of objects of different types.

Under this model, we derive an iterative algorithm, the spectral relational clustering, to cluster the interrelated data objects of different types simultaneously. By iteratively embedding each type of data objects into low-dimensional spaces, the algorithm benefits from the interactions among the hidden structures of data objects of different types. The algorithm has the simplicity of spectral clustering approaches but at the same time also is applicable to relational data with various structures. Theoretic analysis and experimental results demonstrate the promise and effectiveness of the algorithm. We also show that the existing spectral clustering algorithms can be considered as the special cases of the proposed model and algorithm. This provides a unified view to understanding the connections among these algorithms.

1.2.1.1 Model Formulation and Algorithm

In this section, we present a general model for clustering heterogeneous relational data in the spectral domain based on factorizing multiple related matrices.

Given m sets of data objects, $\mathcal{X}_1 = \{x_{11}, \ldots, x_{1n_1}\}, \ldots, \mathcal{X}_m = \{x_{m1}, \ldots, x_{mn_m}\}$, which refer to m different types of objects relating to each other, we are interested in simultaneously clustering $\mathcal{X}_1$ into k_1 disjoint clusters, …, and $\mathcal{X}_m$ into k_m disjoint clusters. We call this task as *collective clustering on heterogeneous relational data*.

To derive a general model for collective clustering, we first formulate the Heterogeneous Relational Data (HRD) as a set of related matrices, in which two matrices are related in the sense that their row indices or column indices refer to the same set of objects. First, if there exist relations between $\mathcal{X}_i$ and $\mathcal{X}_j$ (denoted as $\mathcal{X}_i \sim \mathcal{X}_j$), we represent them as a relation matrix $R^{(ij)} \in \mathbb{R}^{n_i \times n_j}$, where an element $R^{(ij)}_{pq}$ denotes the relation between x_{ip} and x_{jq}. Second, a set of objects $\mathcal{X}_i$ may have its

own features, which could be denoted by a feature matrix $F^{(i)} \in \mathbb{R}^{n_i \times f_i}$, where an element $F^{(i)}_{pq}$ denotes the qth feature values for the object x_{ip} and f_i is the number of features for $\mathcal{X}_i$.

Figure 1.1 shows three examples of the structures of HRD. Example (a) refers to a basic bi-type of relational data denoted by a relation matrix $R^{(12)}$, such as word-document data. Example (b) represents a tri-type of star-structured data, such as Web pages, Web users, and search queries in Web search systems, which are denoted by two relation matrices $R^{(12)}$ and $R^{(23)}$. Example (c) represents the data consisting of shops, customers, suppliers, shareholders, and advertisement media, in which customers (type 5) have features. The data are denoted by four relation matrices $R^{(12)}$, $R^{(13)}$, $R^{(14)}$ and $R^{(15)}$, and one feature matrix $F^{(5)}$.

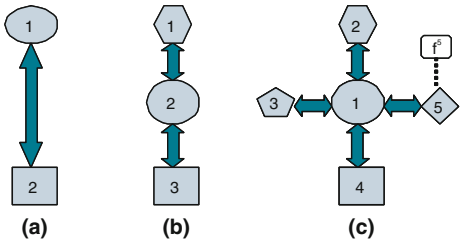


Fig. 1.1 Examples of the structures of the heterogeneous relational data

It has been shown that the hidden structure of a data matrix can be explored by its factorization [13, 39]. Motivated by this observation, we propose a general model for collective clustering, which is based on factorizing the multiple related matrices. In HRD, the cluster structure for a type of objects $\mathcal{X}_i$ may be embedded in multiple related matrices; hence, it can be exploited in multiple related factorizations. First, if $\mathcal{X}_i \sim \mathcal{X}_j$, then the cluster structures of both $\mathcal{X}_i$ and $\mathcal{X}_j$ are reflected in the triple factorization of their relation matrix $R^{(ij)}$ such that $R^{(ij)} \approx C^{(i)}A^{(ij)}(C^{(j)})^T$ [39], where $C^{(i)} \in \{0,1\}^{n_i \times k_i}$ is a *cluster indicator matrix* for $\mathcal{X}_i$ such that $\sum_{q=1}^{k_i} C^{(i)}_{pq} = 1$ and $C^{(i)}_{pq} = 1$ denotes that the pth object in $\mathcal{X}_i$ is associated with the qth cluster. Similarly $C^{(j)} \in \{0,1\}^{n_j \times k_j}$. $A^{(ij)} \in \mathbb{R}^{k_i \times k_j}$ is the *cluster association matrix* such that A^{ij}_{pq} denotes the association between cluster p of $\mathcal{X}_i$ and cluster q of $\mathcal{X}_j$. Second, if $\mathcal{X}_i$ has a feature matrix $F^{(i)} \in \mathbb{R}^{n_i \times f_i}$, the cluster structure is reflected in the factorization of $F^{(i)}$ such that $F^{(i)} \approx C^{(i)}B^{(i)}$, where $C^{(i)} \in \{0,1\}^{n_i \times k_i}$ is a cluster indicator matrix, and $B^{(i)} \in \mathbb{R}^{k_i \times f_i}$ is the feature basis matrix which consists of k_i basis (cluster center) vectors in the feature space.

Based on the above discussions, formally we formulate the task of collective clustering on HRD as the following optimization problem. Considering the most general case, we assume that in HRD, every pair of $\mathcal{X}_i$ and $\mathcal{X}_j$ is related to each other and every $\mathcal{X}_i$ has a feature matrix $F^{(i)}$.

Definition 1 Given m positive numbers $\{k_i\}_{1 \le i \le m}$ and HRD $\{\mathcal{X}_1, \ldots, \mathcal{X}_m\}$, which is described by a set of relation matrices $\{R^{(ij)} \in \mathbb{R}^{n_i \times n_j}\}_{1 \le i < j \le m}$, a set of feature matrices $\{F^{(i)} \in \mathbb{R}^{n_i \times f_i}\}_{1 \le i \le m}$, as well as a set of weights $w_a^{(ij)}$, $w_b^{(i)} \in R_+$ for

different types of relations and features, the task of the collective clustering on the HRD is to minimize

$$L = \sum_{1 \le i < j \le m} w_a^{(ij)} ||R^{(ij)} - C^{(i)} A^{(ij)} (C^{(j)})^T||^2 + \sum_{1 \le i \le m} w_b^{(i)} ||F^{(i)} - C^{(i)} B^{(i)}||^2, \tag{1.1}$$

w.r.t. $C^{(i)} \in \{0, 1\}^{n_i \times k_i}$, $A^{(ij)} \in \mathbb{R}^{k_i \times k_j}$, and $B^{(i)} \in \mathbb{R}^{k_i \times f_i}$ subject to the constraints: $\sum_{q=1}^{k_i} C_{pq}^{(i)} = 1$, where $1 \le p \le n_i$, $1 \le i < j \le m$, and $||\cdot||$ denotes the Frobenius norm for a matrix.

We call the model proposed in Definition 1 as the Collective Factorization on Related Matrices (CFRM).

The CFRM model clusters heterogeneously interrelated data objects simultaneously based on both relation and feature information. The model exploits the interactions between the hidden structures of different types of objects through the related factorizations which share matrix factors, i.e., cluster indicator matrices. Hence, the interactions between hidden structures work in two ways. First, if $\mathcal{X}_i \sim \mathcal{X}_j$, the interactions are reflected as the duality of row clustering and column clustering in $R^{(ij)}$. Second, if two types of objects are indirectly related, the interactions pass along the relation "chains" by a chain of related factorizations, i.e., the model is capable of dealing with influence propagation. In addition to local cluster structure for each type of objects, the model also provides the global structure information by the cluster association matrices, which represent the relations among the clusters of different types of objects.

Based on the CFRM model, we derive an iterative algorithm, called Spectral Relational Clustering (SRC) algorithm [36]. The specific derivation of the algorithm and the proof of the convergence of the algorithm refer to [36]. Further, in Long et al. [36], it is shown that the CFRM model as well as the SRC algorithm is able to handle the general case of heterogeneous relational data, and many existing methods in the literature are either the special cases or variations of this model. Specifically, it is shown that the classic k-means [51], the spectral clustering methods based on graph partitioning [41, 42], and the Bipartite Spectral Graph Partitioning (BSGP) [17, 50] are all the special cases of this general model.

1.2.1.2 Experiments

The SRC algorithm is evaluated on two types of HRD, bi-type relational data and tri-type star-structured data as shown in Fig. 1.1a and b, which represent two basic structures of HRD and arise frequently in real applications.

The data sets used in the experiments are mainly based on the 20 Newsgroups data [33] which contain about 20,000 articles from 20 newsgroups. We pre-process the data by removing stop words and file headers and selecting top 2000 words by the mutual information. The word–document matrix R is based on *tf.idf* and each

document vector is normalized to the unit norm vector. In the experiments the classic k-means is used for initialization and the final performance score for each algorithm is the average of the 20 test runs unless stated otherwise.

Clustering on Bi-type Relational Data

In this section we report experiments on bi-type relational data, word–document data, to demonstrate the effectiveness of SRC as a novel co-clustering algorithm. A representative spectral clustering algorithm, Normalized Cut (NC) spectral clustering [41, 42], and BSGP [17] are used for comparisons.

The graph affinity matrix for NC is $R^T R$, i.e., the cosine similarity matrix. In NC and SRC, the leading k eigenvectors are used to extract the cluster structure, where k is the number of document clusters. For BSGP, the second to the $(\lceil \log_2 k \rceil + 1)$th leading singular vectors are used [17]. k-means is adopted to postprocess the eigenvectors. Before post-processing, the eigenvectors from NC and SRC are normalized to the unit norm vector and the eigenvectors from BSGP are normalized as described by [17]. Since all the algorithms have random components resulting from k-means or itself, at each test we conduct three trials with random initializations for each algorithm and the optimal one provides the performance score for that test run. To evaluate the quality of document clusters, we elect to use the Normalized Mutual Information (NMI) [43], which is a standard measure for the clustering quality.

At each test run, five data sets, multi2 (NG 10, 11), multi3 (NG 1, 10, 20), multi5 (NG 3, 6, 9, 12, 15), multi8 (NG 3, 6, 7, 9, 12, 15, 18, 20), and multi10 (NG 2, 4, 6, 8, 10, 12, 14, 16, 18, 20), are generated by randomly sampling 100 documents from each newsgroup. Here NG i means the ith newsgroup in the original order. For the numbers of document clusters, we use the numbers of the true document classes. For the numbers of word clusters, there are no options for BSGP, since they are restricted to equal to the numbers of document clusters. For SRC, it is flexible to use any number of word clusters. Since how to choose the optimal number of word clusters is beyond the scope of this study, we simply choose one more word cluster than the corresponding document clusters, i.e., 3, 4, 6, 9, and 11. This may not be the best choice but it is good enough to demonstrate the flexibility and effectiveness of SRC.

Figure 1.2a,b, and c show three document embeddings of a multi2 sample, which is sampled from two close newsgroups, *rec.sports.baseball* and *rec.sports.hockey*. In this example, when NC and BSGP fail to separate the document classes, SRC still provides a satisfactory separation. The possible explanation is that the adaptive interactions among the hidden structures of word clusters and document clusters remove the noise to lead to better embeddings. (d) shows a typical run of the SRC algorithm.

Table 1.1 shows NMI scores on all the data sets. We observe that SRC performs better than NC and BSGP on all data sets. This verifies the hypothesis that benefiting from the interactions of the hidden structures of objects with different types, the SRC's adaptive dimensionality reduction has advantages over the dimensionality reduction of the existing spectral clustering algorithms.

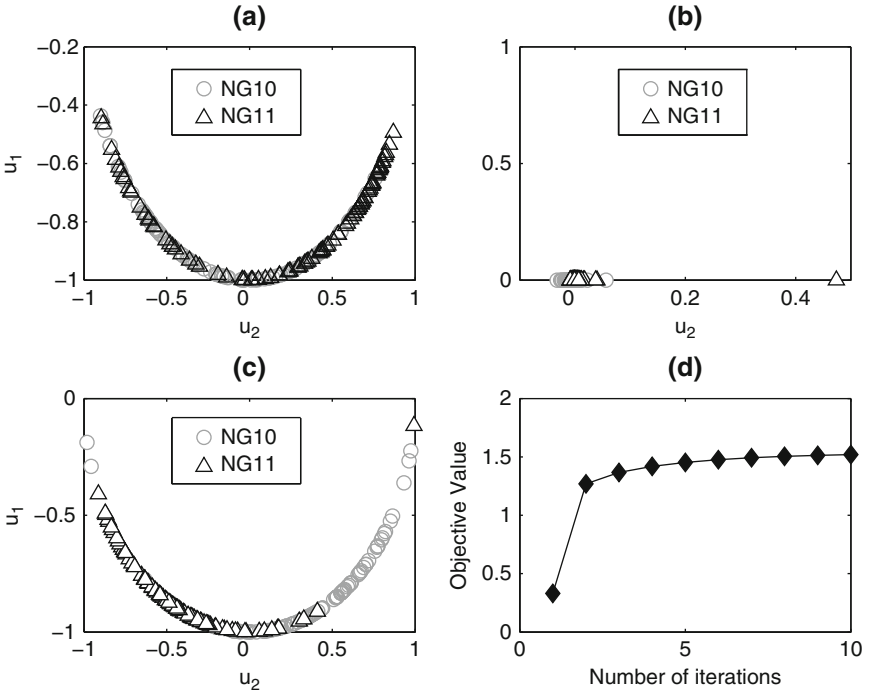


Fig. 1.2 (**a**), (**b**), and (**c**) are document embeddings of multi2 data set produced by NC, BSGP, and SRC, respectively (u_1 and u_2 denote first and second eigenvectors, respectively). (**d**) is an iteration curve for SRC

Table 1.1 NMI comparisons of SRC, NC, and BSGP algorithms

Data set	SRC	NC	BSGP
multi2	0.4979	0.1036	0.1500
multi3	0.5763	0.4314	0.4897
multi5	0.7242	0.6706	0.6118
multi8	0.6958	0.6192	0.5096
multi10	0.7158	0.6292	0.5071

Clustering on Tri-type Relational Data

In this section, we report the experiments on tri-type star-structured relational data to evaluate the effectiveness of SRC in comparison with other two algorithms for HRD clustering. One is based on the m-partite graph partitioning, Consistent Bipartite Graph Co-partitioning (CBGC) [23] (we thank the authors for providing the executable program of CBGC). The other is Mutual Reinforcement K-means (MRK), which is implemented based on the idea of mutual reinforcement clustering.

The first data set is synthetic data, in which two relation matrices, $R^{(12)}$ with 80×100 dimension and $R^{(23)}$ with 100×80 dimension, are binary matrices with 2×2 block structures. $R^{(12)}$ is generated based on the block structure $\left[\begin{smallmatrix} 0.9 & 0.7 \\ 0.8 & 0.9 \end{smallmatrix}\right]$ i.e.,

the objects in cluster 1 of $\mathcal{X}^{(1)}$ is related to the objects in cluster 1 of $\mathcal{X}^{(2)}$ with probability 0.9. $R^{(23)}$ is generated based on the block structure $\left[\begin{smallmatrix} 0.6 & 0.7 \\ 0.7 & 0.6 \end{smallmatrix}\right]$. Each type of objects has two equal size clusters. It is not a trivial task to identify the cluster structure of this data set, since the block structures are subtle. We denote this data set as Binary Relation Matrices (TRM) data.

Other three data sets are built based on the 20 Newsgroups data for hierarchical taxonomy mining and document clustering. In the field of text categorization, hierarchical taxonomy classification is widely used to obtain a better trade-off between effectiveness and efficiency than flat taxonomy classification. To take advantage of hierarchical classification, one must mine a hierarchical taxonomy from the data set. We can see that words, documents, and categories formulate tri-type relational data, which consist of two relation matrices, a word–document matrix $R^{(12)}$, and a document–category matrix $R^{(23)}$ [23].

The true taxonomy structures for the three data sets, TM1, TM2, and TM3, are listed in Table 1.2. For example, TM1 data set is sampled from five categories, in which NG10 and NG11 belong to the same high-level category *res.sports* and NG17, NG18, and NG19 belong to the same high-level category *talk.politics*. Therefore, for the TM1 data set, the expected clustering result on categories should be {NG10, NG11} and {NG17, NG18, NG19} and the documents should be clustered into two clusters according to their categories. The documents in each data set are generated by sampling 100 documents from each category.

Table 1.2 Taxonomy structures for three datasets

Data set	Taxonomy structure
TM1	{NG10, NG11}, {NG17, NG18, NG19}
TM2	{NG2, NG3}, {NG8, NG9}, {NG12, NG13}
TM3	{NG4, NG5}, {NG8, NG9}, {NG14, NG15}, {NG17, NG18}

The number of the clusters used for documents and categories are 2, 3, and 4 for TM1, TM2, and TM3, respectively. For the number of word clusters, we adopt the number of categories, i.e., 5, 6, and 8. For the weights $w_a^{(12)}$ and $w_a^{(23)}$, we simply use equal weight, i.e., $w_a^{(12)} = w_a^{(23)} = 1$. Figure 1.3 illustrates the effects of different weights on embeddings of documents and categories. When $w_a^{(12)} = w_a^{(23)} = 1$, i.e., SRC makes use of both word–document relations and document–category relations, both documents and categories are separated into two clusters very well as in (a) and (b) of Fig. 1.3, respectively; when SRC makes use of only the word–document relations, the documents are separated with partial overlapping as in (c) and the categories are randomly mapped to a couple of points as in (d); when SRC makes use of only the document–category relations, both documents and categories are incorrectly overlapped as in (e) and (f), respectively, since the document–category matrix itself does not provide any useful information for the taxonomy structure.

The performance comparison is based on the cluster quality of documents, since the better it is, the more accurate we can identify the taxonomy structures. Table 1.3 shows NMI comparisons of the three algorithms on the four data sets. The

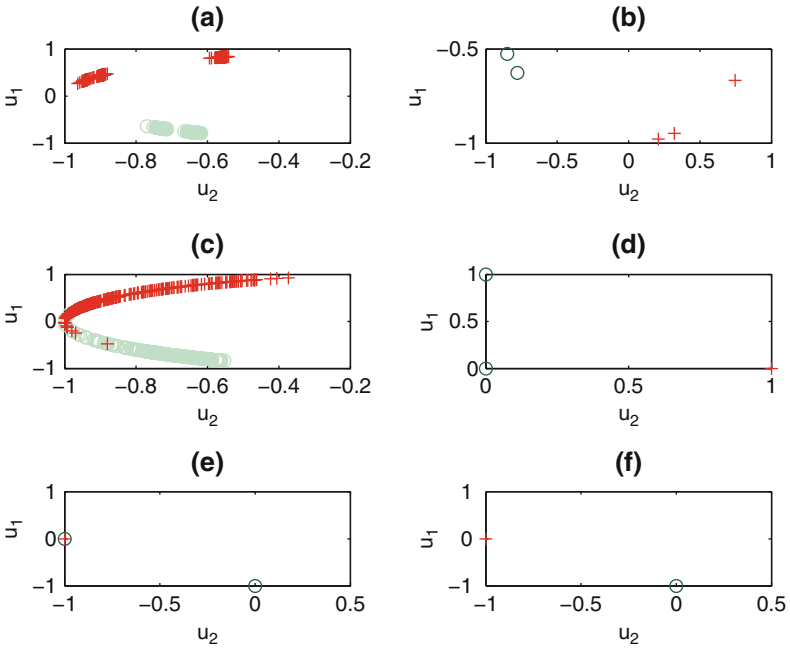


Fig. 1.3 Three pairs of embeddings of documents and categories for the TM1 data set produced by SRC with different weights: (**a**) and (**b**) with $w_a^{(12)} = 1, w_a^{(23)} = 1$; (**c**) and (**d**) with $w_a^{(12)} = 1, w_a^{(23)} = 0$; (**e**) and (**f**) with $w_a^{(12)} = 0, w_a^{(23)} = 1$

Table 1.3 NMI comparisons of SRC, MRK, and CBGC algorithms

Data set	SRC	MRK	CBGC
BRM	0.6718	0.6470	0.4694
TM1	1	0.5243	–
TM2	0.7179	0.6277	–
TM3	0.6505	0.5719	–

NMI score of CBGC is available only for BRM data set because the CBGC program provided by the authors only works for the case of two clusters and small size matrices. We observe that SRC performs better than MRK and CBGC on all data sets. The comparison shows that among the limited efforts in the literature attempting to cluster multi-type interrelated objects simultaneously, SRC is an effective one to identify the cluster structures of HRD.

1.2.2 Homogeneous Relational Clustering Through Convex Coding

The most popular way to solve the problem of clustering the homogeneous relational data such as similarity-based relational data is to formulate it as a graph partitioning

problem, which has been studied for decades. Graph partitioning seeks to cut a given graph into disjoint subgraphs which correspond to disjoint clusters based on a certain edge cut objective. Recently, graph partitioning with an edge cut objective has been shown to be mathematically equivalent to an appropriate weighted kernel k-means objective function [15, 16]. The assumption behind the graph partitioning formulation is that since the nodes within a cluster are similar to each other, they form a dense subgraph. However, in general, this is not true for relational data, i.e., the clusters in relational data are not necessarily *dense* clusters consisting of strongly related objects.

Figure 1.4 shows the relational data of four clusters, which are of two different types. In Fig. 1.4, $\mathcal{C}_1 = \{v_1, v_2, v_3, v_4\}$ and $\mathcal{C}_2 = \{v_5, v_6, v_7, v_8\}$ are two traditional dense clusters within which objects are strongly related to each other. However, $\mathcal{C}_3 = \{v_9, v_{10}, v_{11}, v_{12}\}$ and $\mathcal{C}_4 = \{v_{13}, v_{14}, v_{15}, v_{16}\}$ also form two *sparse* clusters, within which the objects are not related to each other, but they are still "similar" to each other in the sense that they are related to the same set of other nodes. In Web mining, this type of cluster could be a group of music "fans" Web pages which share the same taste on the music and are linked to the same set of music Web pages but are not linked to each other [32]. Due to the importance of identifying this type of clusters (communities), it has been listed as one of the five algorithmic challenges in Web search engines [27]. Note that the cluster structure of the relation data in Fig. 1.4 cannot be correctly identified by graph partitioning approaches, since they look for only dense clusters of strongly related objects by cutting the given graph into subgraphs; similarly, the pure bipartite graph models cannot correctly identify this type of cluster structures. Note that re-defining the relations between the objects (e.g., re-defining 1–0 and 0–1) does not solve the problem in this situation, since there exist both dense and sparse clusters.

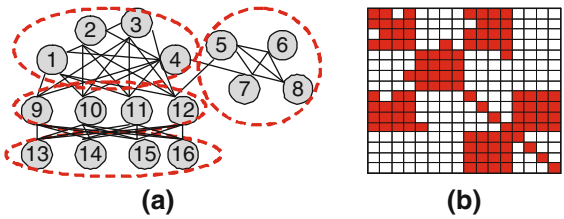


Fig. 1.4 The graph (**a**) and relation matrix (**b**) of the relational data with different types of clusters. In (**b**), the *dark* color denotes 1 and the *light* color denotes 0

If the homogeneous relational data are dissimilarity-based, to apply graph partitioning approaches to them, we need extra efforts on appropriately transforming them into similarity-based data and ensuring that the transformation does not change the cluster structures in the data. Hence, it is desirable for an algorithm to be able to identify the cluster structures no matter which type of relational data is given. This is even more desirable in the situation where the background knowledge about the meaning of the relations is not available, i.e., we are given only a relation matrix and do not know if the relations are similarities or dissimilarities.

In this section, we present a general model for relational clustering based on symmetric convex coding of the relation matrix [35]. The model is applicable to the general homogeneous relational data consisting of only pairwise relations typically without other knowledge; it is capable of learning both dense and sparse clusters at the same time; it unifies the existing graph partition models to provide a generalized theoretical foundation for relational clustering. Under this model, we derive iterative bound optimization algorithms to solve the symmetric convex coding for two important distance functions, Euclidean distance and generalized I-divergence. The algorithms are applicable to general relational data and at the same time they can be easily adapted to learn a specific type of cluster structure. For example, when applied to learning only dense clusters, they provide new efficient algorithms for graph partitioning. The convergence of the algorithms is theoretically guaranteed. Experimental evaluation and theoretical analysis show the effectiveness and great potential of the proposed model and algorithms.

1.2.2.1 Model Formulation and Algorithms

In this section, we describe a general model for homogeneous relational clustering. Let us first consider the relational data in Fig. 1.4. An interesting observation is that although the different types of clusters look so different in the graph from Fig. 1.4a, they all demonstrate block patterns in the relation matrix of Fig. 1.4b (without loss of generality, we arrange the objects from the same cluster together to make the block patterns explicit). Motivated by this observation, we propose the Symmetric Convex Coding (SCC) model to cluster relational data by learning the block pattern of a relation matrix. Since in most applications, the relations are of non-negative values and undirected, homogeneous relational data can be represented as non-negative, symmetric matrices. Therefore, the definition of SCC is given as follows.

Definition 2 Given a symmetric matrix $A \in \mathbb{R}_+$, a distance function $\mathfrak{D}$ and a positive number k, the symmetric convex coding is given by the minimization

$$\min_{\substack{C \in \mathbb{R}_+^{n \times k}, B \in \mathbb{R}_+^{k \times k} \\ C\mathbf{1}=\mathbf{1}}} \mathfrak{D}(A, CBC^T). \tag{1.2}$$

According to Definition 2, the elements of C are between 0 and 1 and the sum of the elements in each row of C equals 1. Therefore, SCC seeks to use the convex combination of the *prototype matrix* B to approximate the original relation matrix. The factors from SCC have intuitive interpretations. The factor C is the soft membership matrix such that C_{ij} denotes the weight that the ith object associates with the jth cluster. The factor B is the prototype matrix such that B_{ii} denotes the connectivity within the ith cluster and B_{ij} denotes the connectivity between the ith cluster and the jth cluster.

SCC provides a general model to learn various cluster structures from relational data. Graph partitioning, which focuses on learning dense cluster structure, can be formulated as a special case of the SCC model. We propose the following theorem

to show that the various graph partitioning objective functions are mathematically equivalent to a special case of the SCC model. Since most graph partitioning objective functions are based on the hard cluster membership, in the following theorem we change the constraints on C as $C \in \mathbb{R}_+$ and $C^T C = I_k$ to make C to be the following cluster indicator matrix,

$$C_{ij} = \begin{cases} \frac{1}{|\pi_j|^{\frac{1}{2}}} & \text{if } v_i \in \pi_j \\ 0 & \text{otherwise,} \end{cases}$$

where $|\pi_j|$ denotes the number of nodes in the jth cluster.

Theorem 1 *The hard version of SCC model under Euclidean distance function and $B = r I_k$ for $r > 0$, i.e.,*

$$\min_{\substack{C \in \mathbb{R}_+^{n \times k}, B \in \mathbb{R}_+^{k \times k} \\ C^T C = I_k}} ||A - C(r I_k) C^T||^2 \tag{1.3}$$

is equivalent to the maximization

$$\max tr(C^T A C), \tag{1.4}$$

where tr denotes the trace of a matrix.

The proof of Theorem 1 may be found in [35].

Theorem 1 states that with the prototype matrix B restricted to be of the form $r I_k$, SCC under Euclidean distance is reduced to the trace maximization in (1.4). Since various graph partitioning objectives, such as ratio association [42], normalized cut [42], ratio cut [8], and Kernighan–Lin objective [31], can be formulated as the trace maximization [15, 16], Theorem 1 establishes the connection between the SCC model and the existing graph partitioning objective functions. Based on this connection, it is clear that the existing graph partitioning models make an implicit assumption for the cluster structure of the relational data, i.e., the clusters are not related to each other (the off-diagonal elements of B are zeroes) and the nodes within clusters are related to each other in the same way (the diagonal elements of B are r). This assumption is consistent with the intuition about the graph partitioning, which seeks to "cut" the graph into k separate subgraphs corresponding to the strongly related clusters.

With Theorem 1 we may put other types of structural constraints on B to derive new graph partitioning models. For example, we fix B as a general diagonal matrix instead of $r I_k$, i.e., the model fixes the off-diagonal elements of B as zero and learns the diagonal elements of B. This is a more flexible graph partitioning model, since it allows the connectivity within different clusters to be different. More generally, we can use B to restrict the model to learn other types of the cluster structures. For example, by fixing diagonal elements of B as zeros, the model focuses on learning only spare clusters (corresponding to bipartite or k-partite subgraphs), which are

important for Web community learning [27, 32]. In summary, the prototype matrix B not only provides the intuition for the cluster structure of the data but also provides a simple way to adapt the model to learn specific types of cluster structures.

Now efficient algorithms for the SCC model may be derived under two popular distance functions, Euclidean distance and generalized I-divergence. SCC under the Euclidean distance, i.e., an algorithm alternatively updating B and C until convergence, is derived and called SCC-ED [35].

If the task is to learn the dense clusters from similarity-based relational data as the graph partitioning does, SCC-ED can achieve this task simply by fixing B as the identity matrix and updating only C until convergence. In other words, these updating rules provide a new and efficient graph partitioning algorithm, which is computationally more efficient than the popular spectral graph partitioning approaches which involve expensive eigenvector computation (typically $O(n^3)$) and the extra post-processing [49] on eigenvectors to obtain the clustering. Compared with the multi-level approaches such as METIS [30], this new algorithm does not restrict clusters to have an equal size.

Another advantage of the SCC-ED algorithm is that it is very easy for the algorithm to incorporate constraints on B to learn a specific type of cluster structures. For example, if the task is to learn the sparse clusters by constraining the diagonal elements of B to be zero, we can enforce this constraint simply by initializing the diagonal elements of B as zeros. Then, the algorithm automatically only updates the off-diagonal elements of B and the diagonal elements of B are "locked" to zeros.

Yet another interesting observation about SCC-ED is that if we set $\alpha = 0$ to change the updating rule for C into the following:

$$C = \tilde{C} \odot \left(\frac{A\tilde{C}B}{\tilde{C}B\tilde{C}^T\tilde{C}B} \right)^{\frac{1}{4}}, \tag{1.5}$$

the algorithm actually provides the symmetric conic coding. This has been touched in the literature as the symmetric case of non-negative factorization [7, 18, 39]. Therefore, SCC-ED under $\alpha = 0$ also provides a theoretically sound solution to the symmetric non-negative matrix factorization.

Under the generalized I-divergence, the SCC objective function is given as follows:

$$D\left(A||CBC^T\right) = \sum_{ij} \left(A_{ij} \log \frac{A_{ij}}{\left[CBC^T\right]_{ij}} - A_{ij} + \left[CBC^T\right]_{ij} \right). \tag{1.6}$$

Similarly, an alternative bound optimization algorithm is derived for this objective function, called SCC-GI [35], which provides another new relational clustering algorithm. Again, when applied to the similarity-based relational data of dense clusters, SCC-GI provides another new and efficient graph partitioning algorithm.

The specific derivation of the two algorithms refers to [35], where the complexity and the convergence issues of the algorithms are discussed.

1.2.2.2 Experiments

This section provides empirical evidence to show the effectiveness of the SCC model and algorithms in comparison with two representative graph partitioning algorithms, a spectral approach, Normalized Cut (NC) [42], and a multi-level algorithm, METIS [30].

Data Sets and Parameter Setting

The data sets used in the experiments include synthetic data sets with various cluster structures and real data sets based on various text data from the 20 Newsgroups [33], WebACE, and TREC [29].

First, we use synthetic binary relational data to simulate homogeneous relational data with different types of clusters such as dense clusters, sparse clusters, and mixed clusters. All the synthetic relational data are generated based on Bernoulli distribution. The distribution parameters to generate the graphs are listed in the second column of Table 1.4 as matrices (true prototype matrices for the data). In a parameter matrix P, P_{ij} denotes the probability that the nodes in the ith cluster are connected to the nodes in the jth cluster. For example, in data set syn3, the nodes in cluster 2 are connected to the nodes in cluster 3 with probability 0.2 and the nodes within a cluster are connected to each other with probability 0. Syn2 is generated by using 1 minus syn1. Hence, syn1 and syn2 can be viewed as a pair of similarity/dissimilarity data. Data set syn4 has 10 clusters mixing with dense clusters and sparse clusters. Due to the space limit, its distribution parameters are omitted here. Totally syn4 has 5000 nodes and about 2.1 million edges.

Table 1.4 Summary of the synthetic relational data

Graph	Parameter	n	k
syn1	$\begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$	900	3
syn2	$1 - syn1$	900	3
syn3	$\begin{bmatrix} 0 & 0.1 & 0.1 \\ 0.1 & 0 & 0.2 \\ 0.1 & 0.2 & 0 \end{bmatrix}$	900	3
syn4	$[0, 1]^{10\times 10}$	5000	10

The graphs based on the text data have been widely used to test graph partitioning algorithms [17, 19, 50]. Note that there also exist feature-based algorithms to directly cluster documents based on word features. However, in this study our focus is on the clustering based on relations instead of features. Hence graph clustering algorithms are used in comparisons. We use various data sets from the 20 Newsgroups [33], WebACE, and TREC [29], which cover data sets of different sizes, different balances, and different levels of difficulties. We construct relational data for each text data set such that objects (documents) are related to each other with cosine similarities between the term-frequency vectors. A summary of all the data sets to construct relational data used in this study is shown in Table 1.5, in which n

Table 1.5 Summary of relational data based on text data sets

Name	n	k	Balance	Source
tr11	414	9	0.046	TREC
tr23	204	6	0.066	TREC
NG17-19	1600	3	0.5	20 Newsgroups
NG1-20	14000	20	1.0	20 Newsgroups
k1b	2340	6	0.043	WebACE
hitech	2301	6	0.192	TREC
classic3	3893	3	0.708	MEDLINE/ CISI/CRANFILD

denotes the number of objects in the relational data, k denotes the number of true clusters, and *balance* denotes the size ratio of the smallest clusters to the largest clusters.

For the number of clusters k, we simply use the number of the true clusters. Note that how to choose the optimal number of clusters is a non-trivial model selection problem and beyond the scope of this study. For performance measure, we elect to use the Normalized Mutual Information (NMI) [43] between the resulting cluster labels and the true cluster labels, which is a standard measure for the clustering quality. The final performance score is the average of 10 runs.

Results and Discussion

Table 1.6 shows the NMI scores of the four algorithms on synthetic and real relational data. Each NMI score is the average of 10 test runs and the standard deviation is also reported. We observe that although there is no single winner on all the data, for most data SCC algorithms perform better than or close to NC and METIS. Especially, SCC-GI provides the best performance on 8 of the 11 data sets.

For the synthetic data set syn1, almost all the algorithms provide perfect NMI score, since the data are generated with very clear dense cluster structures, which can be seen from the parameter matrix in Table 1.4. For data set syn2, the

Table 1.6 NMI comparisons of NC, METIS, SCC-ED, and SCC-GI algorithms (the boldface value indicates the best performance for a given data set)

Data	NC	METIS	SCC-ED	SCC-GI
syn1	0.9652 ± 0.031	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$
syn2	0.8062 ± 0.52	0.000 ± 0.00	0.9038 ± 0.045	$\mathbf{0.9753 \pm 0.011}$
syn3	0.636 ± 0.152	0.115 ± 0.001	0.915 ± 0.145	$\mathbf{1.000 \pm 0.000}$
syn4	0.611 ± 0.032	0.638 ± 0.001	0.711 ± 0.043	$\mathbf{0.788 \pm 0.041}$
tr11	0.629 ± 0.039	0.557 ± 0.001	0.6391 ± 0.033	$\mathbf{0.661 \pm 0.019}$
tr23	0.276 ± 0.023	0.138 ± 0.004	$\mathbf{0.335 \pm 0.043}$	0.312 ± 0.099
NG17-19	0.002 ± 0.002	0.091 ± 0.004	0.1752 ± 0.156	$\mathbf{0.225 \pm 0.045}$
NG1-20	0.510 ± 0.004	$\mathbf{0.526 \pm 0.001}$	0.5041 ± 0.156	0.519 ± 0.010
k1b	0.546 ± 0.021	0.243 ± 0.000	0.537 ± 0.023	$\mathbf{0.591 \pm 0.022}$
hitech	0.302 ± 0.005	$\mathbf{0.322 \pm 0.001}$	0.319 ± 0.012	0.319 ± 0.018
classic3	0.621 ± 0.029	0.358 ± 0.000	0.642 ± 0.043	$\mathbf{0.822 \pm 0.059}$

dissimilarity version of syn1, we use exactly the same set of true cluster labels as that of syn1 to measure the cluster quality; the SCC algorithms still provide almost perfect NMI score; however, METIS totally fails on syn2, since in syn2 the clusters have the form of sparse clusters; and based on the edge cut objective, METIS looks for only dense clusters. An interesting observation is that the NC algorithm does not totally fail on syn2 and in fact it provides a satisfactory NMI score. This is due to that although the original objective of the NC algorithm focuses on dense clusters (its objective function can be formulated as the trace maximization in (1.4)), after relaxing C to an arbitrary orthonormal matrix, what NC actually does is to embed cluster structures into the eigenspace and to discover them by post-processing the eigenvectors. Besides the dense cluster structures, sparse cluster structures could also have a good embedding in the eigenspace under a certain condition.

In data set syn3, the relations within clusters are sparser than the relations between clusters, i.e., it also has sparse clusters, but the structure is more subtle than syn2. We observe that NC does not provide a satisfactory performance and METIS totally fails; in the mean time, SCC algorithms identify the cluster structure in syn3 very well. Data set syn4 is a large relational data set of 10 clusters consisting of four dense clusters and six sparse clusters; we observe that the SCC algorithms perform significantly better than NC and METIS on it, since they can identify both dense clusters and sparse clusters at the same time.

For the real data based on the text data sets, our task is to find dense clusters, which is consistent with the objectives of graph partitioning approaches. Overall, the SCC algorithms perform better than NC and METIS on the real data sets. Especially, SCC-ED provides the best performance in most data sets. The possible reasons for this are discussed as follows. First, the SCC model makes use of any possible block pattern in the relation matrices; on the other hand, the edge-cut-based approaches focus on diagonal block patterns. Hence, the SCC model is more robust to heavily overlapping cluster structures. For example, for the difficult NG17-19 data set, SCC algorithms do not totally fail as NC and METIS do. Second, since the edge weights from different graphs may have very different probabilistic distributions, popular Euclidean distance function, which corresponds to normal distribution assumption, are not always appropriate. By Theorem 1, edge-cut-based algorithms are based on Euclidean distance. On the other hand, SCC-GI is based on generalized I-divergence corresponding to Poisson distribution assumption, which is more appropriate for graphs based on text data. Note that how to choose distance functions for specific graphs is non-trivial and beyond the scope of this study. Third, unlike METIS, the SCC algorithms do not restrict clusters to have an equal size and hence they are more robust to unbalanced clusters.

In the experiments, we observe that SCC algorithms perform stably and rarely provide unreasonable solution, though like other algorithms SCC algorithms provide local optima to the NP-hard clustering problem. In the experiments, we also observe that the order of the actual running time for the algorithms is consistent with theoretical analysis, i.e., METIS$<$SCC$<$NC. For example, in a test run on NG1-20, METIS, SCC-ED, SCC-GI, and NC take 8.96, 11.4, 12.1, and 35.8 s, respectively. METIS is the best, since it is quasi-linear.

We also run the SCC-ED algorithm on the actor/actress graph based on IMDB movie data set for a case study of social network analysis. We formulate a graph of 20,000 nodes, in which each node represents an actors/actresses and the edges denote collaboration between them. The number of the cluster is set to be 200. Although there is no ground truth for the clusters, we observe that the results consist of a large number of interesting and meaningful clusters, such as clusters of actors with a similar style and tight clusters of the actors from a movie or a movie serial. For example, Table 1.7 shows Community 121 consisting of 21 actors/actresses, which contains the actors/actresses in movie series "The Lord of Rings."

Table 1.7 The members of cluster 121 in the actor graph

Cluster 121
Viggo Mortensen, Sean Bean, Miranda Otto, Ian Holm, Brad Dourif, Cate Blanchett, Ian McKellen, Liv Tyler, David Wenham, Christopher Lee, John Rhys-Davies, Elijah Wood, Bernard Hill, Sean Astin, Dominic Monaghan, Andy Serkis, Karl Urban, Orlando Bloom, Billy Boyd, John Noble, Sala Baker

1.3 Generative Approaches to Link-Based Clustering

In this section, we study generative approaches to link-based clustering. Specifically, we present solutions to three different link-based clustering problems, the special homogeneous relational data clustering for documents with citations, the general relational data clustering, and the dynamic relational data clustering.

1.3.1 Special Homogeneous Relational Data—Documents with Citations

One of the most popular scenarios for link-based clustering is document clustering. Here textual documents form a special case of the general homogeneous relational data scenario, in which a document links to another one through a citation. In this section, we showcase how to use a generative model, a specific topic model, to solve for the document clustering problem.

By capturing the essential characteristics in documents, one gives documents a new representation, which is often more parsimonious and less noise-sensitive. Among the existing methods that extract essential characteristics from documents, topic model plays a central role. Topic models extract a set of latent topics from a corpus and as a consequence represent documents in a new latent semantic space. One of the well-known topic models is the Probabilistic Latent Semantic Indexing (PLSI) model proposed by Hofmann [28]. In PLSI each document is modeled

as a probabilistic mixture of a set of topics. Going beyond PLSI, Blei et al. [5] presented the Latent Dirichlet Allocation (LDA) model by incorporating a prior for the topic distributions of the documents. In these probabilistic topic models, one assumption underpinning the generative process is that the documents are independent. However, this assumption does not always hold true in practice, because documents in a corpus are usually related to each other in certain ways. Very often, one can explicitly observe such relations in a corpus, e.g., through the citations and co-authors of a paper. In such a case, these observations should be incorporated into topic models in order to derive more accurate latent topics that better reflect the relations among the documents.

In this section, we present a generative model [24] called the *citation-topic* (CT) model for modeling linked documents that explicitly considers the relations among documents. In this model, the content of each document is a mixture of two sources: (1) the topics of the given document and (2) the topics of the documents that are related to (e.g., cited by) the given document. This perspective actually reflects the process of writing a scientific article: the authors probably first learn knowledge from the literature and then combine their own creative ideas with the learned knowledge to form the content of the paper. Furthermore, to capture the indirect relations among documents, CT contains a generative process to select related documents where the related documents are not necessarily directly linked to the given document. CT is applied to the document clustering task and the experimental comparisons against several state-of-the-art approaches that demonstrate very promising performances.

1.3.1.1 Model Formulation and Algorithm

Suppose that the corpus consists of N documents $\{d_j\}_{j=1}^{N}$ in which M distinct words $\{w_i\}_{i=1}^{M}$ occur. Each document d might have a set of citations C_d, and thus the documents are linked together by these citations.

CT assumes the following generative process for each word w in the document d in the corpus.

1. Choose a related document c from $p(c|d, \boldsymbol{\Xi})$, a multinomial probability conditioned on the document d.
2. Choose a topic z from the topic distribution of the document c, $p(z|c, \boldsymbol{\Theta})$.
3. Choose a word w which follows the multinomial distribution $p(w|z, \boldsymbol{\Psi})$ conditioned on the topic z.

As a result, one obtains the observed pair (d, w), while the latent random variables c, z are discarded. To obtain a document d, one repeats this process $|d|$ times, where $|d|$ is the length of the document d. The corpus is obtained once every document in the corpus is generated by this process, as shown in Fig. 1.5. In this generative model, the dimensionality K of the topic variable z is assumed known and the document relations are parameterized by an $N \times N$ matrix $\boldsymbol{\Xi}$ where

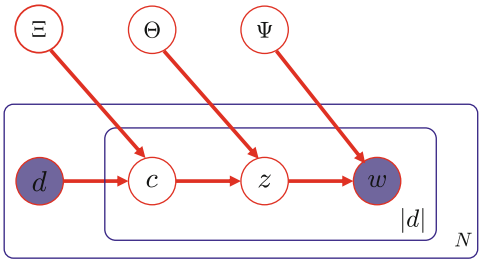


Fig. 1.5 CT using the plate notation

$\Xi_{lj} = p(c = l|d = j)$, which is computed from the citation information of the corpus.

Following the maximum likelihood principle, one estimates the parameters by maximizing the log-likelihood function

$$\mathcal{L} = \sum_{j=1}^{N} \sum_{i=1}^{M} n(w_i, d_j) \log p(w_i|d_j), \tag{1.7}$$

where $n(w_i, d_j)$ denotes the number of the times w_i occurs in d_j. According to the above generative process, the log-likelihood function can be rewritten as the following equation

$$\mathcal{L} = \sum_{j=1}^{N} \sum_{i=1}^{M} n(w_i, d_j) \log \left\{ \sum_{l=1}^{K} \sum_{h=1}^{N} p(w_i|z_l) p(z_l|d_h) p(d_h|d_j) \right\}. \tag{1.8}$$

The expectation–maximization (EM) algorithm can be applied to estimate the parameters.

The document relation matrix $\boldsymbol{\Xi}$ is computed from the citation information of the corpus. Suppose that the document d_j has a set of citations Q_{d_j}. A matrix $\mathbf{S}$ is constructed to denote the direct relationships among the documents as follows: $S_{lj} = 1/|Q_{d_j}|$ for $d_l \in Q_{d_j}$ and 0 otherwise, where $|Q_{d_j}|$ denotes the size of the set Q_{d_j}. A simple method to obtain $\boldsymbol{\Xi}$ is to set $\boldsymbol{\Xi} = \mathbf{S}$. However, this strategy only captures *direct* relations among the documents and overlooks *indirect* relationships. To better capture this transitive property, we choose a related document by a random walk on the directed graph represented by $\mathbf{S}$. The probability that the random walk stops at the current node (and therefore chooses the current document as the related document) is specified by a parameter α. According to the properties of random walk, $\boldsymbol{\Xi}$ can be obtained by $\boldsymbol{\Xi} = (1 - \alpha)(\mathbf{I} - \alpha\mathbf{S})^{-1}$. The specific algorithm refers to [24].

1.3.1.2 Experiments

The experimental evaluations are reported on the document clustering task for a standard data set Cora with the citation information available. Cora [40] contains

the papers published in the conferences and journals of the different research areas in computer science, such as artificial intelligence, information retrieval, and hardware. A unique label has been assigned to each paper to indicate the research area it belongs to. These labels serve as the ground truth in our performance studies. In the Cora data set, there are 9998 documents where 3609 distinct words occur.

By representing documents in terms of latent topic space, topic models can assign each document to the most probable latent topic according to the topic distributions of the documents. For the evaluation purpose, CT is compared with the following representative clustering methods.

1. Traditional K-means.
2. Spectral Clustering with Normalized Cuts (Ncut) [42].
3. Non-negative Matrix Factorization (NMF) [48].
4. Probabilistic Latent Semantic Indexing (PLSI) [28].
5. Latent Dirichlet Allocation (LDA) [5].
6. PHITS [11].
7. PLSI+PHITS, which corresponds to $\alpha = 0.5$ in [12].

The same evaluation strategy is used as in [48] for the clustering performance. The test data used for evaluating the clustering methods are constructed by mixing the documents from multiple clusters randomly selected from the corpus. The evaluations are conducted for different numbers of clusters K. At each run of the test, the documents from a selected number K of clusters are mixed, and the mixed document set, along with the cluster number K, is provided to the clustering methods. For each given cluster number K, 20 test runs are conducted on different randomly chosen clusters, and the final performance scores are obtained by averaging the scores over the 20 test runs.

The parameter α is simply fixed at 0.99 for the CT model. The accuracy comparisons with various numbers of clusters are reported in Fig. 1.6, which shows that CT has the best performance in terms of the accuracy and the relationships among the documents do offer help in the document clustering.

1.3.2 General Relational Clustering Through a Probabilistic Generative Model

In this section, as another example of a generative model in machine learning, we present a probabilistic generative framework to the general relational clustering. As mentioned before, in general, relational data contain three types of information, attributes for individual objects, homogeneous relations between objects of the same type, and heterogeneous relations between objects of different types. For example, for a scientific publication relational data set of papers and authors, the personal information such as affiliation for authors is the attributes; the citation relations among papers are homogeneous relations; the authorship relations between papers and authors are heterogeneous relations. Such data violate the

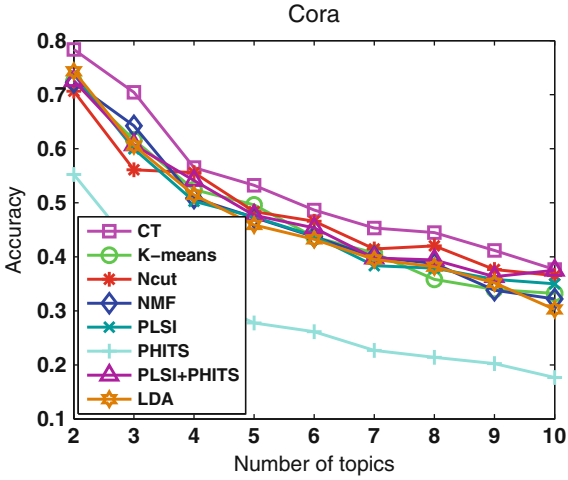


Fig. 1.6 Accuracy comparisons (the higher, the better)

classic IID assumption in machine learning and statistics and present huge challenges to traditional clustering approaches. In Section 1.2.1, we have also shown that an intuitive solution to transform relational data into flat data and then to cluster each type of objects independently may not work. Moreover, a number of important clustering problems, which have been of intensive interest in the literature, can be viewed as special cases of the general relational clustering. For example, graph clustering (partitioning) [6, 8, 19, 26, 30, 42] can be viewed as clustering on single-type relational data consisting of only homogeneous relations (represented as a graph affinity matrix); co-clustering [1, 14] which arises in important applications such as document clustering and micro-array data clustering can be formulated as clustering on bi-type relational data consisting of only heterogeneous relations. Recently, semi-supervised clustering [3, 45] has attracted significant attention, which is a special type of clustering using both labeled and unlabeled data. In [37], it is shown that semi-supervised clustering can be formulated as clustering on single-type relational data consisting of attributes and homogeneous relations.

Therefore, relational data present not only huge challenges to traditional unsupervised clustering approaches but also great need for theoretical unification of various clustering tasks. In this section, we present a probabilistic framework for general relational clustering [37], which also provides a principal framework to unify various important clustering tasks including traditional attribute-based clustering, semi-supervised clustering, co-clustering, and graph clustering. The framework seeks to identify cluster structures for each type of data objects and interaction patterns between different types of objects. It is applicable to relational data of various structures. Under this framework, two parametric hard and soft relational clustering algorithms are developed under a large number of exponential family distributions. The algorithms are applicable to various relational data from various applications and at

the same time unify a number of state-of-the-art clustering algorithms: co-clustering algorithms, the k-partite graph clustering, Bregman k-means, and semi-supervised clustering based on hidden Markov random fields.

1.3.2.1 Model Formulation and Algorithms

With different compositions of three types of information, attributes, homogeneous relations, and heterogeneous relations, relational data could have very different structures. Figure 1.7 shows three examples of the structures of relational data. Figure 1.7a refers to a simple bi-type of relational data with only heterogeneous relations such as word–document data. Figure 1.7b represents bi-type data with all types of information, such as actor–movie data, in which actors (type 1) have attributes such as gender; actors are related to each other by collaboration in movies (homogeneous relations); and actors are related to movies (type 2) by taking roles in movies (heterogeneous relations). Figure 1.7c represents the data consisting of companies, customers, suppliers, shareholders, and advertisement media, in which customers (type 5) have attributes.

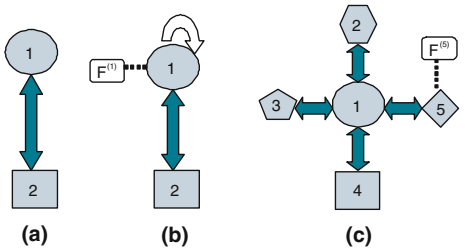


Fig. 1.7 Examples of the structures of relational data

In this study, a relational data set is represented as a set of matrices. Assume that a relational data set has m different types of data objects, $\mathcal{X}^{(1)} = \{x_i^{(1)}\}_{i=1}^{n_1}, \ldots, \mathcal{X}^{(m)} = \{x_i^{(m)}\}_{i=1}^{n_m}$, where n_j denotes the number of objects of the jth type and $x_p^{(j)}$ denotes the name of the pth object of the jth type. The observations of the relational data are represented as three sets of matrices, attribute matrices $\{\mathbf{F}^{(j)} \in \mathbb{R}^{d_j \times n_j}\}_{j=1}^{m}$, where d_j denotes the dimension of attributes for the jth type objects and $\mathbf{F}_{\cdot p}^{(j)}$ denotes the attribute vector for object $x_p^{(j)}$; homogeneous relation matrices $\{\mathbf{S}^{(j)} \in \mathbb{R}^{n_j \times n_j}\}_{j=1}^{m}$, where $\mathbf{S}_{pq}^{(j)}$ denotes the relation between $x_p^{(j)}$ and $x_q^{(j)}$; heterogeneous relation matrices $\{\mathbf{R}^{(ij)} \in \mathbb{R}^{n_i \times n_j}\}_{i,j=1}^{m}$, where $\mathbf{R}_{pq}^{(ij)}$ denotes the relation between $x_p^{(i)}$ and $x_q^{(j)}$. The above representation is a general formulation. In real applications, not every type of objects has attributes, homogeneous relations, and heterogeneous relations all together. For example, the relational data set in Fig. 1.7a is represented by only one heterogeneous matrix $\mathbf{R}^{(12)}$, and the one in Fig. 1.7b is represented by three matrices, $\mathbf{F}^{(1)}$, $\mathbf{S}^{(1)}$, and $\mathbf{R}^{(12)}$. Moreover, for a

specific clustering task, we may not use all available attributes and relations after feature or relation selection pre-processing.

Mixed membership models, which assume that each object has mixed membership denoting its association with classes, have been widely used in the applications involving soft classification [20], such as matching words and pictures [5], race genetic structures [5, 46], and classifying scientific publications [21]. Consequently, a relational mixed membership model is developed to cluster relational data (which is referred to *mixed membership relational clustering* or MMRC throughout the rest of the section).

Assume that each type of objects $\mathcal{X}^{(j)}$ has k_j latent classes. We represent the membership vectors for all the objects in $\mathcal{X}^{(j)}$ as a membership matrix $\Lambda^{(j)} \in [0, 1]^{k_j \times n_j}$ such that the sum of elements of each column $\Lambda^{(j)}_{\cdot p}$ is 1 and $\Lambda^{(j)}_{\cdot p}$ denotes the membership vector for object $x^{(j)}_p$, i.e., $\Lambda^{(j)}_{gp}$ denotes the probability that object $x^{(j)}_p$ associates with the gth latent class. We also write the parameters of distributions to generate attributes, homogeneous relations, and heterogeneous relations in matrix forms. Let $\Theta^{(j)} \in \mathbb{R}^{d_j \times k_j}$ denote the distribution parameter matrix for generating attributes $\mathbf{F}^{(j)}$ such that $\Theta^{(j)}_{\cdot g}$ denotes the parameter vector associated with the gth latent class. Similarly, $\Gamma^{(j)} \in \mathbb{R}^{k_j \times k_j}$ denotes the parameter matrix for generating homogeneous relations $\mathbf{S}^{(j)}$; $\Upsilon^{(ij)} \in \mathbb{R}^{k_i \times k_j}$ denotes the parameter matrix for generating heterogeneous relations $\mathbf{R}^{(ij)}$. In summary, the parameters of MMRC model are

$$\Omega = \left\{ \{\Lambda^{(j)}\}_{j=1}^m, \{\Theta^{(j)}\}_{j=1}^m, \{\Gamma^{(j)}\}_{j=1}^m, \{\Upsilon^{(ij)}\}_{i,j=1}^m \right\}.$$

In general, the meanings of the parameters, Θ, Λ, and Υ, depend on the specific distribution assumptions. However, in [37], it is shown that for a large number of exponential family distributions, these parameters can be formulated as expectations with intuitive interpretations.

Next, we introduce the latent variables into the model. For each object x^j_p, a latent cluster indicator vector is generated based on its membership parameter $\Lambda^{(j)}_{\cdot p}$, which is denoted as $\mathbf{C}^{(j)}_{\cdot p}$, i.e., $\mathbf{C}^{(j)} \in \{0, 1\}^{k_j \times n_j}$ is a latent indicator matrix for all the jth type objects in $\mathcal{X}^{(j)}$.

Finally, we present the generative process of observations, $\{\mathbf{F}^{(j)}\}_{j=1}^m$, $\{\mathbf{S}^{(j)}\}_{j=1}^m$, and $\{\mathbf{R}^{(ij)}\}_{i,j=1}^m$ as follows:

1. For each object $x^{(j)}_p$
   - Sample $\mathbf{C}^{(j)}_{\cdot p} \sim Multinomial\left(\Lambda^{(j)}_{\cdot p}, 1\right)$.
2. For each object $x^{(j)}_p$
   - Sample $\mathbf{F}^{(j)}_{\cdot p} \sim \Pr\left(\mathbf{F}^{(j)}_{\cdot p} | \Theta^{(j)} \mathbf{C}^{(j)}_{\cdot p}\right)$.

3. For each pair of objects $x_p^{(j)}$ and $x_q^{(j)}$
   - Sample $\mathbf{S}_{pq}^{(j)} \sim \Pr\left(\mathbf{S}_{pq}^{(j)} | \left(\mathbf{C}_{\cdot p}^{(j)}\right)^T \Gamma^{(j)} \mathbf{C}_{\cdot q}^{(j)}\right)$.
4. For each pair of objects $x_p^{(i)}$ and $x_q^{(j)}$
   - Sample $\mathbf{R}_{pq}^{(ij)} \sim \Pr\left(\mathbf{R}_{pq}^{(ij)} | \left(\mathbf{C}_{\cdot p}^{(i)}\right)^T \Upsilon^{(ij)} \mathbf{C}_{\cdot q}^{(j)}\right)$.

In the above generative process, a latent indicator vector for each object is generated based on multinomial distribution with the membership vector as parameters. Observations are generated independently conditioning on latent indicator variables. The parameters of condition distributions are formulated as products of the parameter matrices and latent indicators, i.e., $\Pr\left(\mathbf{F}_{\cdot p}^{(j)} | \mathbf{C}_{\cdot p}^{(j)}, \Theta^{(j)}\right) = \Pr\left(F_{\cdot p}^{(j)} | \Theta^{(j)} \mathbf{C}_{\cdot p}^{(j)}\right)$, $\Pr\left(\mathbf{S}_{pq}^{(j)} | \mathbf{C}_{\cdot p}^{(j)}, \mathbf{C}_{\cdot q}^{(j)}, \Gamma^{(j)}\right) = \Pr\left(\mathbf{S}_{pq}^{(j)} | \left(\mathbf{C}_{\cdot p}^{(j)}\right)^T \Gamma^{(j)} \mathbf{C}_{\cdot q}^{(j)}\right)$, and $\Pr(\mathbf{R}_{pq}^{(ij)} | \mathbf{C}_{\cdot p}^{(i)}, \mathbf{C}_{\cdot q}^{(j)}, \Upsilon^{(ij)}) = \Pr\left(\mathbf{R}_{pq}^{(ij)} | \left(\mathbf{C}_{\cdot p}^{(i)}\right)^T \Upsilon^{(ij)} \mathbf{C}_{\cdot q}^{(j)}\right)$. Under this formulation, an observation is sampled from the distributions of its associated latent classes. For example, if $\mathbf{C}_{\cdot p}^{(i)}$ indicates that $x_p^{(i)}$ is with the gth latent class and $\mathbf{C}_{\cdot q}^{(j)}$ indicates that $x_q^{(j)}$ is with the hth latent class, then $\left(\mathbf{C}_{\cdot p}^{(i)}\right)^T \Upsilon^{(ij)} \mathbf{C}_{\cdot q}^{(j)} = \Upsilon_{gh}^{(ij)}$. Hence, we have $\Pr\left(\mathbf{R}_{pq}^{(ij)} | \Upsilon_{gh}^{(ij)}\right)$ implying that the relation between $x_p^{(i)}$ and $x_q^{(j)}$ is sampled by using the parameter $\Upsilon_{gh}^{(ij)}$.

With matrix representation, the joint probability distribution over the observations and the latent variables can be formulated as follows:

$$\Pr(\Psi|\Omega) = \prod_{j=1}^{m} \Pr\left(\mathbf{C}^{(j)} | \Lambda^{(j)}\right) \prod_{j=1}^{m} \Pr\left(\mathbf{F}^{(j)} | \Theta^{(j)} \mathbf{C}^{(j)}\right) \prod_{j=1}^{m} \Pr\left(\mathbf{S}^{(j)} | \left(\mathbf{C}^{(j)}\right)^T \Gamma^{(j)} \mathbf{C}^{(j)}\right) \prod_{i=1}^{m} \prod_{j=1}^{m} \Pr\left(\mathbf{R}^{(ij)} | \left(\mathbf{C}^{(i)}\right)^T \Upsilon^{(ij)} \mathbf{C}^{(j)}\right), \tag{1.9}$$

where $\Psi = \left\{\{\mathbf{C}^{(j)}\}_{j=1}^m, \{\mathbf{F}^{(j)}\}_{j=1}^m, \{\mathbf{S}^{(j)}\}_{j=1}^m, \{\mathbf{R}^{(ij)}\}_{i,j=1}^m\right\}$,

$\Pr\left(\mathbf{C}^{(j)} | \Lambda^{(j)}\right) = \prod_{p=1}^{n_j} Multinomial\left(\Lambda_{\cdot p}^{(j)}, 1\right)$,

$\Pr\left(\mathbf{F}^{(j)} | \Theta^{(j)} \mathbf{C}^{(j)}\right) = \prod_{p=1}^{n_j} \Pr\left(\mathbf{F}_{\cdot p}^{(j)} | \Theta^{(j)} \mathbf{C}_{\cdot p}^{(j)}\right)$,

$\Pr\left(\mathbf{S}^{(j)} | \left(\mathbf{C}^{(j)}\right)^T \Gamma^{(j)} \mathbf{C}^{(j)}\right) = \prod_{p,q=1}^{n_j} \Pr\left(\mathbf{S}_{pq}^{(j)} | \left(\mathbf{C}_{\cdot p}^{(j)}\right)^T \Gamma^{(j)} \mathbf{C}_{\cdot q}^{(j)}\right)$,

and similarly for $\mathbf{R}^{(ij)}$.

Based on the MMRC model, we are able to derive the soft version MMRC, the hard version MMRC, as well as the mixed version MMRC (i.e., the combination of the soft version and the hard version MMRC) algorithms under all the exponential family functions [37]. In addition, we also show that many existing models and algorithms in the literature are the variations or special cases of the MMRC model. Specifically, we have demonstrated this unified view to the classic attribute-based clustering (including the k-means), the mixture model EM clustering, semi-supervised clustering, co-clustering, and graph clustering in the literature.

1.3.2.2 Experiments

This section provides empirical evidence to show the effectiveness of the MMRC model and algorithms. Since a number of state-of-the-art clustering algorithms [1–3, 10, 14, 34] can be viewed as special cases of the MMRC model and algorithms, the experimental results in these efforts also illustrate the effectiveness of the MMRC model and algorithms. Here we apply MMRC algorithms to the tasks of graph clustering, bi-clustering, tri-clustering, and clustering on a general relational data set of all three types of information. In the experiments, we use mixed version MMRC, i.e., hard MMRC initialization followed by soft MMRC. Although MMRC can adopt various distribution assumptions, due to space limit, we use MMRC under normal or Poisson distribution assumption in the experiments. However, this does not imply that they are optimal distribution assumptions for the data. How to decide the optimal distribution assumption is beyond the scope of this study.

For performance measure, we elect to use the Normalized Mutual Information (NMI) [43] between the resulting cluster labels and the true cluster labels, which is a standard way to measure the cluster quality. The final performance score is the average of 10 runs.

Graph Clustering

In this section, we present experiments on the MMRC algorithm under normal distribution in comparison with two representative graph partitioning algorithms, the spectral graph partitioning (SGP) from [41] that is generalized to work with both normalized cut and ratio association, and the classic multi-level algorithm, METIS [30].

The graphs based on the text data have been widely used to test graph partitioning algorithms [17, 19, 50]. In this study, we use various data sets from the 20 Newsgroups [33], WebACE, and TREC [29], which cover data sets of different sizes, different balances, and different levels of difficulties. The data are pre-processed by removing the stop words and each document is represented by a term-frequency vector using TF-IDF weights. Then we construct relational data for each text data set such that objects (documents) are related to each other with the cosine similarities between the term-frequency vectors. A summary of all the data sets to construct relational data used in this study is shown in Table 1.8, in which n denotes the

Table 1.8 Summary of relational data for graph clustering

Name	n	k	Balance	Source
tr11	414	9	0.046	TREC
tr23	204	6	0.066	TREC
NG1-20	14000	20	1.0	20 Newsgroups
k1b	2340	6	0.043	WebACE

number of objects in the relational data, k denotes the number of true clusters, and *balance* denotes the size ratio of the smallest clusters to the largest clusters.

For the number of clusters k, we simply use the number of the true clusters. Note that how to choose the optimal number of clusters is a non-trivial model selection problem and beyond the scope of this study.

Figure 1.8 shows the NMI comparison of the three algorithms. We observe that although there is no single winner on all the graphs, overall the MMRC algorithm performs better than SGP and METIS. Especially on the difficult data set tr23, MMRC increases the performance about 30%. Hence, MMRC under normal distribution provides a new graph partitioning algorithm which is viable and competitive compared with the two existing state-of-the-art graph partitioning algorithms. Note that although the normal distribution is most popular, MMRC under other distribution assumptions may be more desirable in specific graph clustering applications depending on the statistical properties of the graphs.

Bi-clustering and Tri-clustering

In this section, we apply the MMRC algorithm under Poisson distribution to clustering bi-type relational data, word–document data, and tri-type relational data,

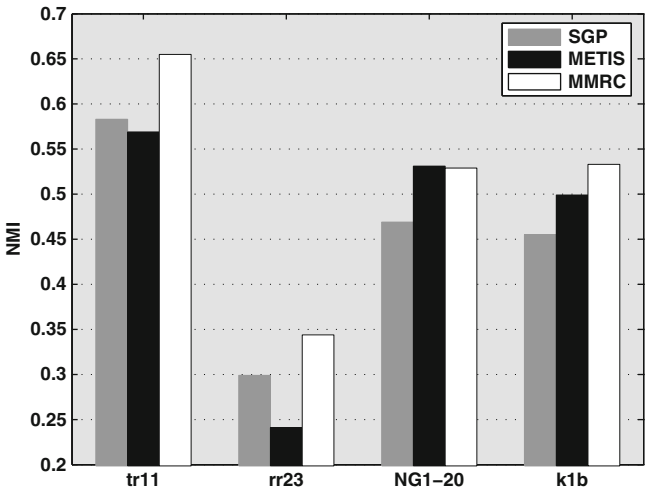


Fig. 1.8 NMI comparison of SGP, METIS, and MMRC algorithms

word–document–category data. Two algorithms, Bipartite Spectral Graph partitioning (BSGP) [17] and Relation Summary Network under Generalized I-divergence (RSN-GI) [38], are used as comparison in bi-clustering. For tri-clustering, Consistent Bipartite Graph Co-partitioning (CBGC) [23] and RSN-GI are used as comparison.

The bi-type relational data, word–document data, are constructed based on various subsets of the 20-Newsgroups data. We pre-process the data by selecting the top 2000 words by the mutual information. The document–word matrix is based on *tf.idf* weighting scheme and each document vector is normalized to a unit L_2 norm vector. Specific details of the data sets are listed in Table 1.9. For example, for the data set *BT-NG3* we randomly and evenly sample 200 documents from the corresponding newsgroups; then we formulate a bi-type relational data set of 1600 documents and 2000 words.

Table 1.9 Subsets of the 20-Newsgroups data for the bi-type relational data

Dataset Name	Newsgroups Included	Number of documents per group	Total number of documents
BT-NG1	rec.sport.baseball, rec.sport.hockey	200	400
BT-NG2	comp.os.ms-windows.misc, comp.windows.x, rec.motorcycles, sci.crypt, sci.space	200	1000
BT-NG3	comp.os.ms-windows.misc, comp.windows.x, misc.forsale, rec.motorcycles,rec.motorcycles,sci.crypt, sci.space, talk.politics.mideast, talk.religion.misc	200	1600

The tri-type relational data are built based on the 20 Newsgroups data for hierarchical taxonomy mining. In the field of text categorization, hierarchical taxonomy classification is widely used to obtain a better trade-off between effectiveness and efficiency than flat taxonomy classification. To take advantage of hierarchical classification, one must mine a hierarchical taxonomy from the data set. We see that words, documents, and categories formulate a sandwich structure tri-type relational data set, in which documents are the central-type nodes. The links between documents and categories are constructed such that if a document belongs to k categories, the weights of links between this document and these k category nodes are $1/k$ (refer to [23] for details). The true taxonomy structures for the two data sets, *TP-TM1* and *TP-TM2*, are documented in Table 1.10.

Table 1.10 Taxonomy structures of the two data sets for constructing tri-partite relational data

Dataset	Taxonomy structure
TT-TM1	{rec.sport.baseball, rec.sport.hockey}, {talk.politics.guns, talk.politics.mideast, talk.politics.misc}
TT-TM2	{comp.graphics, comp.os.ms-windows.misc}, {rec.autos, rec.motorcycles}, {sci.crypt, sci.electronics}

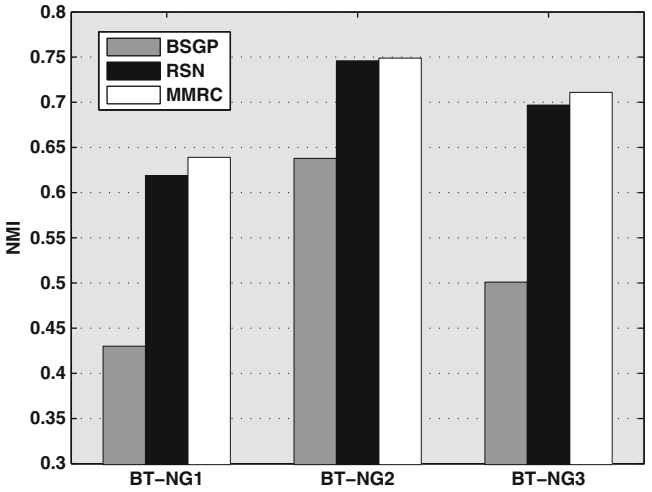


Fig. 1.9 NMI comparison of BSGP, RSN, and MMRC algorithms for bi-type data

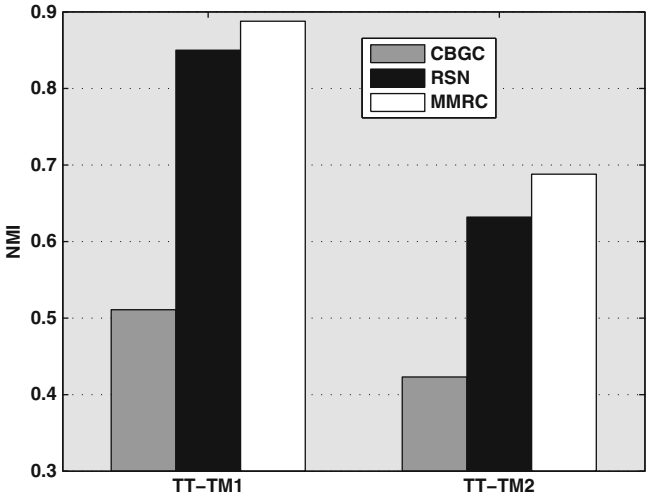


Fig. 1.10 NMI comparison of CBGC, RSN, and MMRC algorithms for tri-type data

Figures 1.9 and 1.10 show the NMI comparison of the three algorithms on bi-type and tri-type relational data, respectively. We observe that the MMRC algorithm performs significantly better than BSGP and CBGC. MMRC performs slightly better than RSN on some data sets. Since RSN is a special case of hard MMRC, this shows that mixed MMRC improves hard MMRC's performance on the data sets. Therefore, compared with the existing state-of-the-art algorithms, the MMRC algorithm performs more effectively on these bi-clustering or tri-clustering tasks and on

the other hand, it is flexible for different types of multi-clustering tasks which may be more complicated than tri-type clustering.

A Case Study on Actor–movie Data

We also run the MMRC algorithm on the actor–movie relational data based on the IMDB movie data set for a case study. In the data, actors are related to each other by collaborations (homogeneous relations); actors are related to movies by taking roles in the movies (heterogeneous relations); movies have attributes such as release time and rating (note that there are no links between movies). Hence the data have all the three types of information. We formulate a data set of 20,000 actors and 4000 movies. We run experiments with $k = 200$. Although there is no ground truth for the data's cluster structure, we observe that most resulting clusters are actors or movies of a similar style such as actions or tight groups from specific movie serials. For example, Table 1.11 shows cluster 23 of actors and cluster 118 of movies; the parameter $\Upsilon_{23,118}$ shows that these two clusters are strongly related to each other. In fact, the actor cluster contains the actors in the movie series "The Lord of the Rings." Note that if we only have one type of actor objects, we only get the actor clusters, but with two types of nodes, although there is no link between the movies, we also get the related movie clusters to explain how the actors are related.

Table 1.11 Two clusters from actor–movie data

Cluster 23 of actors
Viggo Mortensen, Sean Bean, Miranda Otto, Ian Holm, Christopher Lee, Cate Blanchett, Ian McKellen, Liv Tyler, David Wenham, Brad Dourif, John Rhys-Davies, Elijah Wood, Bernard Hill, Sean Astin, Andy Serkis, Dominic Monaghan, Karl Urban, Orlando Bloom, Billy Boyd, John Noble, Sala Baker
Cluster 118 of movies
The Lord of the Rings: The Fellowship of the Ring (2001) The Lord of the Rings: The Two Towers (2002) The Lord of the Rings: The Return of the King (2003)

1.3.3 Dynamic Relational Data Clustering Through Graphical Models

We have studied extensively on static relational data clustering in the previous sections. In this section, we switch our focus to dynamic scenarios. One popular example of the dynamic scenarios is the evolutionary clustering. Evolutionary clustering is a recently identified new and hot research topic in data mining. Evolutionary clustering addresses the evolutionary trend development regarding a collection of

data items that evolves over the time. From time to time, with the evolution of the data collection, new data items may join the collection and existing data items may leave the collection; similarly, from time to time, cluster structure and cluster number may change during the evolution. Due to the nature of the evolution, model selection must be solved as part of a solution to the evolutionary clustering problem at each time. Consequently, evolutionary clustering poses a greater challenge than the classic, static clustering problem as many existing solutions to the latter problem typically assume that the model selection is still an open problem in the clustering literature.

In evolutionary clustering, one of the most difficult and challenging issues is to solve the correspondence problem. The correspondence problem refers to the correspondence between different local clusters across the times due to the evolution of the distribution of the clusters, resulting in cluster–cluster correspondence and cluster transition correspondence issues. All the existing methods in the literature fail to address the correspondence problems explicitly.

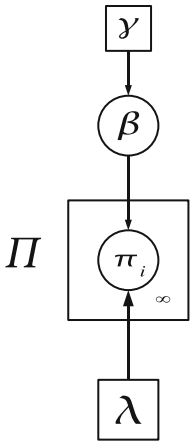
On the other hand, solutions to the evolutionary clustering problem have found a wide spectrum of applications for trend development analysis, social network evolution analysis, and dynamic community development analysis. Potential and existing applications include daily news analysis to observe news focus change, blog analysis to observe community development, and scientific publications analysis to identify the new and hot research directions in a specific area. Consequently, evolutionary clustering has recently become a very hot and focused research topic.

In this study [47], we show a new statistical graphical model HDP-HTM that we have developed as an effective solution to the evolutionary clustering problem. In this new model, we assume that the cluster structure at each time is a mixture model of the clusters for the data collection at that time; in addition, clusters at different times may share common clusters, resulting in explicitly addressing the cluster–cluster correspondence issue. we adopt the Hierarchical Dirichlet Processes (HDP) [44] with a set of common clusters at the top level of the hierarchy and the local clusters at the lower level at different times sharing the top-level clusters. Further, data and clusters evolve over the time with new clusters and new data items possibly joining the collection and with existing clusters and data items possibly leaving the collection at different times, leading to the cluster structure and the number of clusters evolving over the time. Here, we use the state transition matrix to explicitly reflect the cluster-to-cluster transitions between different times, resulting an explicitly effective solution to the cluster transition correspondence issue. Consequently, we propose the Infinite Hierarchical Hidden Markov State model (iH^{2}MS) to construct the Hierarchical Transition Matrix (HTM) at different times to capture the cluster-to-cluster transition evolution.

1.3.3.1 Infinite Hierarchical Hidden Markov State Model (iH^{2}MS)

Here, we present a new infinite hierarchical Hidden Markov State model (iH^{2}MS) for Hierarchical Transition Matrix (HTM) and provide an update construction scheme based on this model. Figure 1.11 illustrates this model.

Fig. 1.11 The iH^{2}MS model



Traditionally, Hidden Markov model (HMM) has a *finite* state space with K hidden states, say $\{1, 2, \ldots K\}$. For the hidden state sequence $\{s_1, s_2, \ldots, s_T\}$ up to time T, there is a K by K state transition probability matrix Π governed by Markov dynamics with all the elements $\pi_{i,j}$ of each row $\boldsymbol{\pi}_i$ summed to 1:

$$\pi_{i,j} = p(s_t = j | s_{t-1} = i).$$

The initial state probability for state i is $p(s_1 = i)$ with the summation of all the initial probabilities equal to 1. For observation x_t in the observation sequence $\{x_1, x_2, \ldots, x_T\}$, given state $s_t \in \{1, 2, \ldots, K\}$, there is a parameter ϕ_{s_t} drawn from the base measure H which parameterizes the observation likelihood probability:

$$x_t | s_t \sim F(\phi_{s_t}).$$

However, when dealing with a countable *infinite* state space, $\{1, 2, \ldots K, \ldots\}$, we must adopt a new model similar to that in [4] for a state transition probability matrix with an *infinite* matrix dimension. Thus, the dimension of the state transition probability matrix now has become infinite. $\boldsymbol{\pi}_i$, the ith row of the transition probability matrix Π, may be represented as the mixing proportions for all the next infinite states, given the current state. Thus, we model it as a Dirichlet process (DP) with an infinite dimension with the summation of all the elements in a row normalized to 1, which leads to an infinite number of DPs' construction for an infinite transition probability matrix.

With no further prior knowledge on the state sequence, a typical prior for the transition probability may be the symmetric Dirichlet distributions. Similar to [44], we intend to construct a hierarchical Dirichlet model to keep different rows of the transition probability matrix to share part of the prior mixing proportions of each

state at the top level. Consequently, we adopt a new state model, Infinite Hierarchical Hidden Markov State model (iH^{2}MS), to construct the Infinite Transition Probability Matrix which is called the Hierarchical Transition Matrix (HTM).

Similar to HDP [44], we draw a random probability measure on the infinite state space β as the top level prior from $stick(\gamma)$ represented as the mixing proportions of each state:

$$\beta = (\beta_k)_{k=1}^{\infty} \quad \beta_k = \beta_k{}' \prod_{l=1}^{k-1}(1-\beta_l{}') \quad \beta_k{}' \sim Beta(1,\gamma). \tag{1.10}$$

Here, the mixing proportion of state k, β_k, may also be interpreted as the prior mean of the transition probabilities leading to state k. Hence, β may be represented as the prior random measure of a transition probability DP.

For the ith row of the transition matrix Π, π_i, we sample it from $DP(\lambda, \beta)$ with a smaller concentration parameter λ implying a larger variability around the mean measure β. The stick-breaking representation for π_i is as follows:

$$\pi_i = (\pi_{i,k})_{k=1}^{\infty} \quad \pi_{i,k} = \pi_{i,k}{}' \prod_{l=1}^{k-1}(1-\pi_{i,l}{}') \quad \pi_{i,k}{}' \sim Beta(1,\lambda). \tag{1.11}$$

Specifically, $\pi_{i,k}$ is the state transition probability from the previous state i to the current state k as $p(s_t = k|s_{t-1} = i)$.

Now, each row of the transition probability matrix is represented as a DP which shares the same reasonable prior on the mixing proportions of the states. For a new row corresponding to a new state k, we simply draw a transition probability vector π_k from $DP(\lambda, \beta)$, resulting in constructing a countably infinite transition probability matrix. The transition probability constructed by iH^{2}MS may be further extended to the scenario where there are more than one state at each time [47]. HTM is estimated through the maximum likelihood principle [47].

1.3.3.2 Model Formulation and Algorithm

To capture the state (cluster) transition correspondence during the evolution at different times, we have proposed the HTM; at the same time, we must capture the state–state (cluster–cluster) correspondence, which may be handled by a hierarchical model with the top level corresponding to the global states[1] and the lower level corresponding to the local states, where it is natural to model the statistical process as HDP [44]. Consequently, we intend to combine HDP with HTM as a new HDP-HTM model, as illustrated in Fig. 1.12.

[1] Each state is represented as a distinct cluster.

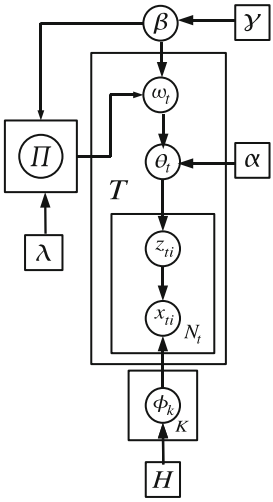


Fig. 1.12 The HDP-HTM model

Let the global state space S denote the global cluster set, which includes all the states $S_t \subseteq S$ at all the times t. The global observation set X includes all the observations X_t at each time t, of which each data item i is denoted as $x_{t,i}$.

We draw the global mixing proportion from the global states β with the stick-breaking representation using the concentration parameter γ from (1.10). The global measure G_0 may be represented as

$$G_0 = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k},$$

where ϕ_k is drawn from the base probability measure H with pdf h, and δ_{ϕ_k} is the concentration measure on ϕ_k.

Different from HDP, here we must consider the evolution of the data and the states (i.e., the clusters). The distribution of the clusters at time t is not only governed by the global measure G_0 but also controlled by the data and cluster evolution in the history. Consequently, we make an assumption that the data and the clusters at time t are generated from the previous data and clusters, according to the mixture proportions of each cluster and the transition probability matrix. The global prior mixture proportions for the clusters are β, and the state transition matrix Π provides the information of the previous state evolution in the history up to time t. Now, the expected number of the data items generated by cluster k is proportional to the number of data items in the clusters in the history multiplied by the transition probabilities from these clusters to state k; specifically, the mean mixture proportion for cluster k at time t, ω_t, is defined as follows:

$$\omega_{t,k} = \sum_{j=1}^{\infty} \beta_j \pi_{j,k}.$$

More precisely, ω_t is further obtained by

$$\omega_t = \beta \cdot \Pi. \tag{1.12}$$

Clearly, by the transition probability property, $\sum_{k=1}^{\infty} \omega_{t,k} = 1$, $\sum_{k=1}^{\infty} \pi_{i,k} = 1$, and the stick-breaking property $\sum_{j=1}^{\infty} \beta_j = 1$:

$$\sum_{k=1}^{\infty} \omega_{t,k} = \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} \beta_j \pi_{j,k} = \sum_{j=1}^{\infty} \beta_j \sum_{k=1}^{\infty} \pi_{j,k} = \sum_{j=1}^{\infty} \beta_j = 1.$$

Thus, the mean mixture proportion ω_t may be taken as the new probability measure at time t on the global cluster set. With the concentration parameter α, we draw the mixture proportion vector θ_t from $\mathrm{DP}(\alpha, \omega_t)$

$$\theta_t | \alpha, \omega_t \sim \mathrm{DP}(\alpha, \omega_t).$$

Now, at time t, the local measure G_t shares the global clusters parameterized by $\phi = (\phi_k)_{k=1}^{\infty}$ with the mixing proportion vector θ_t.

$$G_t = \sum_{k=1}^{\infty} \theta_{t,k} \delta_{\phi_k}$$

At time t, given the mixture proportion of the clusters θ_t, we draw a cluster indicator $z_{t,i}$ for data item $x_{t,i}$ from a multinomial distribution:

$$z_{t,i} | \theta_t \sim \mathrm{Mult}(\theta_t)$$

Once we have the cluster indicator $z_{t,i}$, data item $x_{t,i}$ may be drawn from distribution F with pdf f, parameterized by ϕ from the base measure H.

$$x_{t,i} | z_{t,i}, \phi \sim f(x | \phi_{z_{t,i}})$$

Finally, we summarize the data generation process for HDP-HTM as follows.

1. Sample the cluster parameter vector ϕ from the base measure H. The number of the parameters is unknown a priori, but is determined by the data when a new cluster is needed.
2. Sample the global cluster mixture vector β from $stick(\gamma)$.
3. At time t, compute the mean measure ω_t for the global cluster set by β and Π according to (1.12).

4. At time t, sample the local mixture proportion θ_t by DP(α, ω_t).
5. At time t, sample the cluster indicator $z_{t,i}$ from $Mult(\theta_t)$ for data item $x_{t,i}$.
6. At time t, sample data item $x_{t,i}$ from $f(x|\phi_{z_{t,i}})$ given cluster indicator $z_{t,i}$ and parameter vector ϕ.

1.3.3.3 Experiments

We have evaluated the HDP-HTM model in an extensive scale against the state-of-the-art literature. We compare HDP-HTM in performance with evolutionary spectral clustering PCM and PCQ algorithms [9] and HDP [44] for the synthetic data and the real data in the application of document evolutionary clustering; for the experiments in text data evolutionary clustering, we have also evaluated the HDP-HTM model in comparison with LDA [5, 25] in addition. In particular, the evaluations are performed in three data sets, a synthetic data set, the 20 Newsgroups data set, and a Google daily news data set we have collected over a period of 5 continuous days.

Synthetic Data set

We have generated a synthetic data set in a scenario of evolutionary development. The data are a collection of mixture models with the number of the clusters unknown a priori with a smooth transition over the time during the evolution. Specifically, we simulate the scenario of the evolution over 10 different times with each time's collection according to a DP mixture model with 200 two-dimensional Gaussian distribution points. Ten Gaussian points in $\mathbf{N}(\mathbf{0}, \mathbf{2I})$ are set as the 10 global clusters' mean parameters. Then 200 Gaussian points within a cluster are sampled with this cluster's mean parameter and deviation parameter sampling from $\mathbf{N}(\mathbf{0}, \mathbf{0.2I})$, where $\mathbf{I}$ is the identify matrix. After the generation of such a data set, we obtain the number of the clusters and the cluster assignments as the ground truth. We intentionally generate different numbers of the clusters at different times, as shown in Fig. 1.15.

In the inference process, we tune the hyperparameters as follows. In each iteration, we use the vague Gamma priors [22] to update α, λ, and γ from $\Gamma(1, 1)$. Figure 1.13 shows an example of the clustering results between HDP-HTM and PCQ at time 8 for the synthetic data. Clearly, HDP-HTM has a much better performance than PCQ in these synthetic data.

For a more systematic evaluation on this synthetic data set, we use NMI (Normalized Mutual Information) [43] to quantitatively compare the clustering performances among all the four algorithms (HDP-HTM, HDP, PCM, and PCQ). NMI measures how much information two random distribution variables (computed clustering assignment and ground truth clustering assignment) share, the larger the better with 1 as normalized maximum value. Figure 1.14 documents the performance comparison. From this figure, the average NMI values across 10 times for HDP-HTM and HDP are 0.86 and 0.78, respectively, while those for PCQ and PCM are 0.70 and 0.71, respectively. HDP works worse than HDP-HTM for the synthetic data. The reason is that HDP model is unable to capture the cluster transition correspondence during the evolution among the data collections across the time in this case while

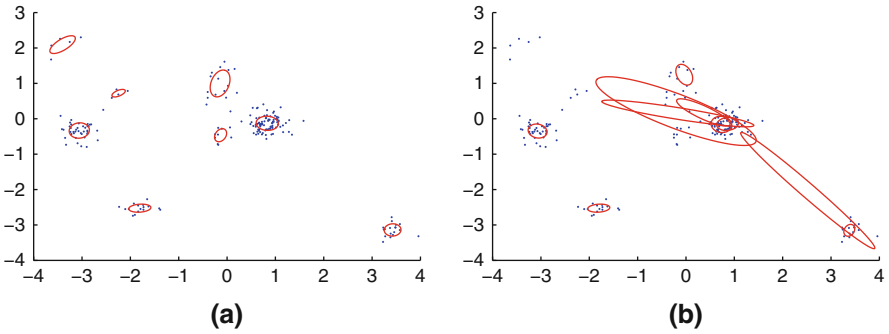


Fig. 1.13 Illustrated clustering results of HDP-HTM (**a**) and PCQ (**b**) for the synthetic data

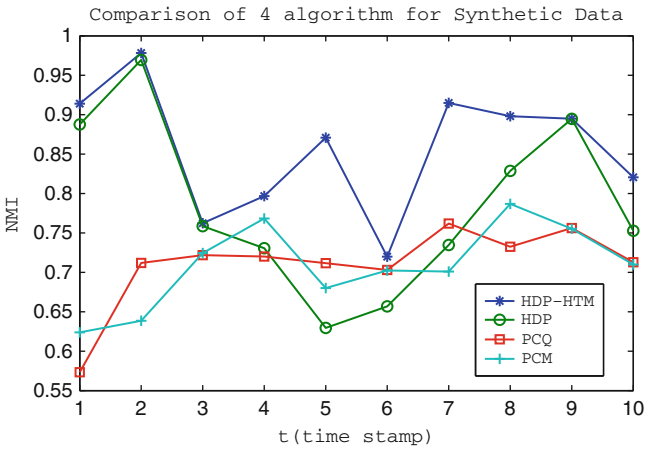


Fig. 1.14 The NMI performance comparison of the four algorithms on the synthetic data set

HDP-HTM is able to explicitly solve for this correspondence problem; on the other hand, HDP still performs better than PCQ and PCM as HDP is able to learn the cluster number automatically during the evolution.

Since one of the advantages of the HDP-HTM model is to be able to learn the number of the clusters and the clustering structures during the evolution, we report this performance for HDP-HTM compared with HDP on this synthetic data set in Fig. 1.15. Here, we define the expected number of the clusters at each time as the average number of the clusters in all the posterior sampling iterations after the burn-in period. Thus, these numbers are not necessarily integers. Clearly, both models are able to learn the cluster numbers, with HDP-HTM having a better performance than HDP. Since both PCQ and PCM do not have this capability, they are not included in this evaluation.

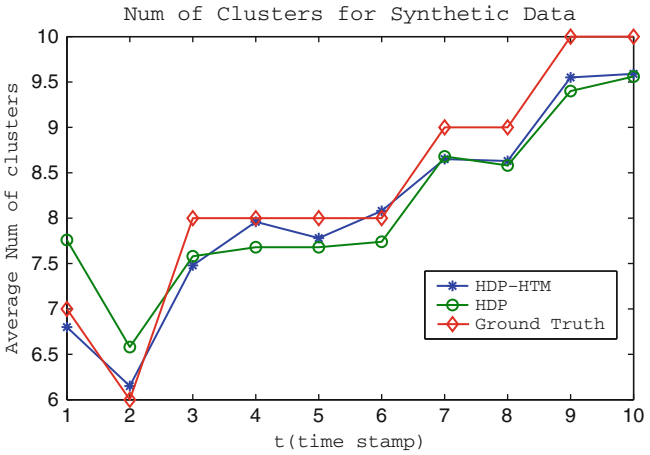


Fig. 1.15 The cluster number learning performance of the HDP-HTM in comparison with HDP on the synthetic data set

Real Data Set

In order to showcase the performance of HDP-HTM model on real data applications, we apply HDP-HTM to a subset of the 20 Newsgroups data.[2] We intentionally set the number of the clusters at each time as the same number to accommodate the comparing algorithms PCQ and PCM which have this assumption of the same cluster number over the evolution. Also we select 10 clusters (i.e., topics) from the data set (alt.atheism, comp.graphics, rec.autos, rec.sport.baseball, sci.crypt, sci.electronics, sci.med, sci.space, soc.religion.christian, talk.politics.mideast), with each having 100 documents. To "simulate" the corresponding 5 different times, we then split the data set into 5 different collections, each of which has 20 documents randomly selected from the clusters. Thus, each collection at a time has 10 topics to generate words. We have pre-processed all the documents with the standard text processing for removing the stop words and stemming the remaining words.

To apply the HDP-HTM and HDP models, a symmetric Dirichlet distribution is used with the parameter 0.5 for the prior base distribution H. In each iteration, we update α, γ, and λ in HDP-HTM, from the gamma priors $\Gamma(0.1,\ 0.1)$. For LDA, α is set 0.1 and the prior distribution of the topics on the words is a symmetric Dirichlet distribution with concentration parameter 1. Since LDA only works for one data collection and requires a known cluster number in advance, we explicitly apply LDA to the data collection with the ground truth cluster number as input at each time.

Figure 1.16 reports the overall performance comparison among all the five methods using NMI metric again. Clearly HDP-HTM outperforms PCQ, PCM, HDP,

[2] http:kdd.ics.uci.edu/databases/20newsgroups/

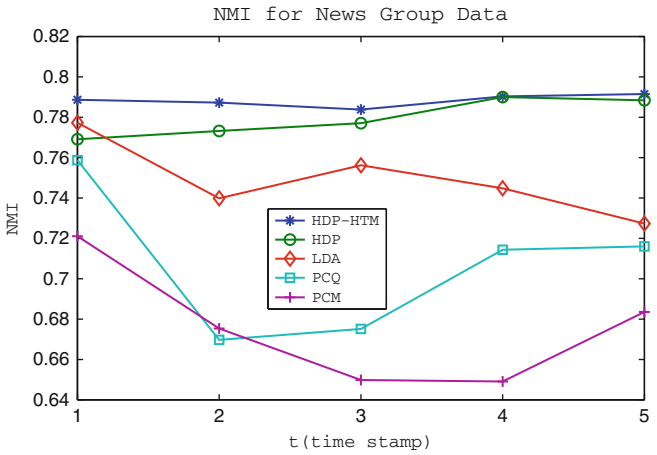


Fig. 1.16 The NMI performance comparison among the five algorithms on the 20 Newsgroups data set

and LDA at all the times; in particular, the difference is substantial for PCQ and PCM. Figure 1.17 further reports the performance on learning the cluster numbers at different times for HDP-HTM compared with HDP. Both models have a reasonable performance in automatically learning the cluster number at each time in comparison with the ground truth, with HDP-HTM having a clearly better performance than HDP in average.

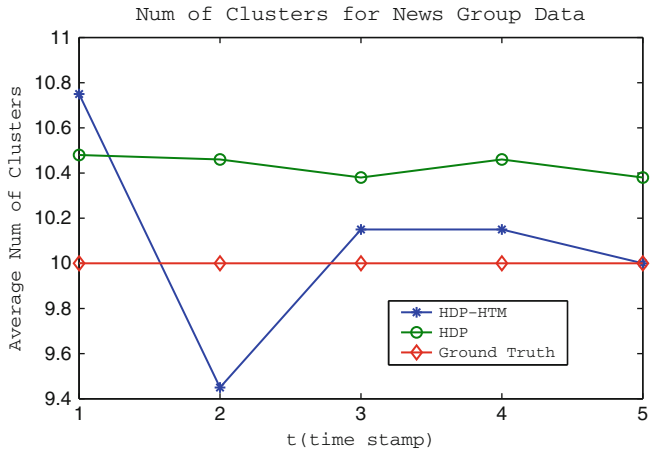


Fig. 1.17 Cluster number learning performance of HDP-HTM in comparison with HDP on the 20 Newsgroups data set

In order to truly demonstrate the performance of HDP-HTM in comparison with the state-of-the-art literature on a real evolutionary clustering scenario, we have manually collected Google News articles for a continuous period of 5 days with both the data items (i.e., words in the articles) and the clusters (i.e., the news topics) evolving over the time. The evolutionary ground truth for this data set is as follows. For each of the continuous 5 days, we have the number of the words, the number of the clusters, the number of the documents as (6113, 5, 50), (6356, 6, 60), (7063, 5, 50), (7762, 6, 60), and (8035, 6, 60), respectively. In order to accommodate the assumption of PCM and PCQ that the cluster number stays the same during the evolution, but at the same time in order to demonstrate the capability of HDP-HTM to automatically learn the cluster number at each evolutionary time, we intentionally set the news topic number (i.e., the cluster number) at each day's collection to have a small variation deviation during the evolution. Again, in order to compare the text clustering capability of LDA [5, 25] with a known topic number in advance, we use the ground truth cluster number at each time as the input to LDA. The parameter tuning process is similar to that in the experiment using the 20 Newsgroups data set.

Figure 1.18 reports the NMI-based performance evaluations among the five algorithms. Again, HDP-HTM outperforms PCQ, PCM, HDP, and LDA at all the times, especially substantially better than PCQ, PCM, and LDA. PCQ and PCM fail completely in most of the cases as they assume that the number of the clusters remains the same during the evolution, which is not true in this scenario.

Figure 1.19 further reports the performance on learning the cluster numbers for different times for HDP-HTM compared with HDP. In this data set, HDP-HTM has a much better performance than HDP to learn the cluster numbers automatically at all the times.

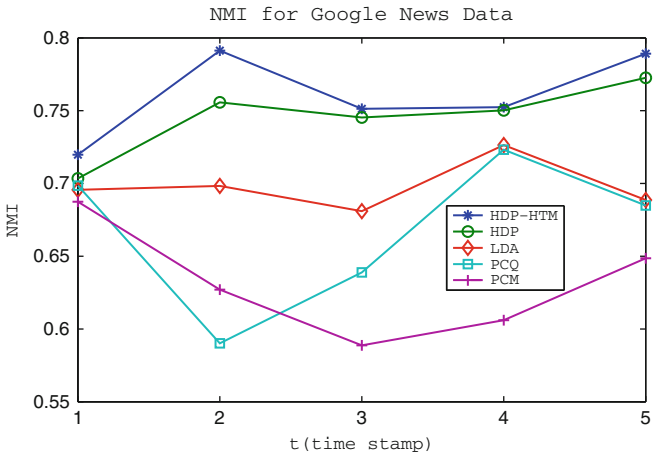


Fig. 1.18 The NMI performance comparison for all the five algorithms on the Google News data set

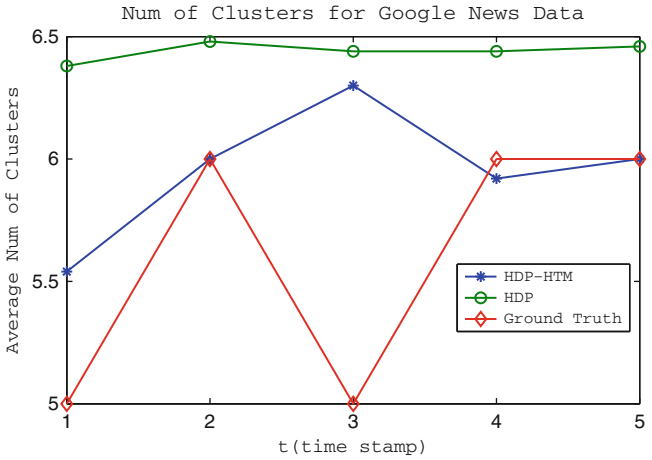


Fig. 1.19 The cluster number learning performance of HDP-HTM in comparison with HDP on the Google News data set

1.4 Conclusions

In this chapter, we have reviewed several specific machine learning techniques used for different categories of link-based or relational data clustering in two paradigms — deterministic approaches and generative approaches. Specifically, we have showcased a spectral clustering technique for heterogeneous relational clustering, a symmetric convex coding technique for homogeneous relational clustering, a citation model for the special homogeneous relational clustering — clustering textual documents with citations, a probabilistic generative model for general relational clustering, as well as a statistical graphical model for dynamic relational clustering. All these machine learning approaches are based on the mathematical foundation of matrix computation theory, probability, and statistics.

Acknowledgments This work is supported in part through NSF grants [IIS-0535162, IIS-0812114, IIS-0905215, and DBI-0960443], as well as graduate research internships at Google Research Labs and NEC Laboratories America, Inc. Yun Chi, Yihong Gong, Xiaoyun Wu, and Shenghuo Zhu have made contributions to part of this material.

References

1. A. Banerjee, I. S. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha. A generalized maximum entropy approach to bregman co-clustering and matrix approximation. In *KDD*, pages 509–514, 2004.
2. A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.
3. S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings ACM KDD04*, pages 59–68, Seattle, WA, August 2004.

4. M. J. Beal, Z. Ghahramani, and C. E. Rasmussen. The infinite hidden markov model. In *NIPS 14*, 2002.
5. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 993–1022, 2003.
6. T. N. Bui and C. Jones. A heuristic for reducing fill-in in sparse matrix factorization. In *PPSC*, pages 445–452, 1993.
7. M. Catral, L. Han, M. Neumann, and R. J. Plemmons. On reduced rank nonnegative matrix factorization for symmetric nonnegative matrices. *Linear Algebra and Its Application*, 2004.
8. P. K. Chan, M. D. F. Schlag, and J. Y. Zien. Spectral k-way ratio-cut partitioning and clustering. In *DAC'93*, pages 749–754, 1993.
9. Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 153–162, 2007.
10. H. Cho, I. Dhillon, Y. Guan, and S. Sra. Minimum sum squared residue co-clustering of gene expression data. In *SDM*, 2004.
11. D. Cohn and H. Chang. Learning to probabilistically identify authoritative documents. In *Proceeding of ICML*, pages 167–174, 2000.
12. D. A. Cohn and T. Hofmann. The missing link – a probabilistic model of document content and hypertext connectivity. In *Proceedings of NIPS*, pages 430–436, 2000.
13. D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.
14. I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *KDD'03*, pages 89–98, 2003.
15. I. Dhillon, Y. Guan, and B. Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical Report TR-04-25, University of Texas at Austin, 2004.
16. I. Dhillon, Y. Guan, and B. Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD'05*, 2005.
17. I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*, pages 269–274, 2001.
18. C. Ding, X. He, and H. D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *SDM'05*, 2005.
19. C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of ICDM 2001*, pages 107–114, 2001.
20. E. Erosheva and S. E. Fienberg. Bayesian mixed membership models for soft clustering and classification. *Classification-The Ubiquitous Challenge*, pages 11–26, 2005.
21. E.A. Erosheva, S.E. Fienberg, and J. Lafferty. Mixed membership models of scientific publications. In *NAS*.
22. M. D. Escobar and M. West. Bayesian density estimation and inference using mixtures. *The Annals of Statistics*, 90:577–588, 1995.
23. B. Gao, T. Y. Liu, X. Zheng, Q. S. Cheng, and W. Y. Ma. Consistent bipartite graph copartitioning for star-structured high-order heterogeneous data co-clustering. In *KDD'05*, pages 41–50, 2005.
24. Z. Guo, S. Zhu, Y. Chi, Z. Zhang, and Y. Gong. A latent topic model for linked documents. In *Proceedings of ACM SIGIR*, 2009.
25. G. Heinrich. Parameter estimation for text analysis. *Technical Report*, 2004.
26. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing '95*, page 28, 1995.
27. M. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1573–1579, 2003.
28. T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings SIGIR*, pages 50–57, 1999.
29. G. Karypis. A clustering toolkit, 2002.
30. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

31. B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.
32. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks*, 31(11–16), 1999.
33. K. Lang. News weeder: Learning to filter netnews. In *ICML*, 1995.
34. T. Li. A general model for clustering binary data. In *KDD'05*, 2005.
35. B. Long, Z. Zhang, and P. S. Yu. Relational clustering by symmetric convex coding. In *Proceedings of International Conference on Machine Learning*, 2007.
36. B. Long, Z. Zhang, X. Wu, and P. S. Yu. Spectral clustering for multi-type relational data. In *Proceedings of ICML*, 2006.
37. B. Long, Z. Zhang, and P. S. Yu. A probabilistic framework for relational clustering. In *Proceedings of ACM KDD*, 2007.
38. B. Long, X. Wu, Z. Zhang, and P. S. Yu. Unsupervised learning on k-partite graphs. In *KDD-2006*, 2006.
39. B. Long, Z. M. Zhang, and P. S. Yu. Co-clustering by block value decomposition. In *KDD'05*, 2005.
40. A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.
41. A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, 2001.
42. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 22(8):888–905, 2000.
43. A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining partitionings. In *AAAI 2002*, pages 93–98, 2002.
44. Y. Teh, M. Beal M. Jordan, and D. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2007.
45. K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *ICML-2001*, pages 577–584, 2001.
46. E. P. Xing, A. Y. Ng, M. I. Jorda, and S. Russel. Distance metric learning with applications to clustering with side information. In *NIPS'03*, volume 16, 2003.
47. T. Xu, Z. Zhang, P. S. Yu, and B. Long. Evolutionary clustering by hierarchical dirichlet process with hidden markov state. In *Proceedings of IEEE ICDM*, 2008.
48. W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of SIGIR*, pages 267–273, 2003.
49. S. Yu and J. Shi. Multiclass spectral clustering. In *ICCV'03*, 2003.
50. H. Zha, C. Ding, M. Gu, X. He, and H. Simon. Bi-partite graph partitioning and data clustering. In *ACM CIKM'01*, 2001.
51. H. Zha, C. Ding, M. Gu, X. He, and H. Simon. Spectral relaxation for k-means clustering. *Advances in Neural Information Processing Systems*, 14, 2002.

Chapter 2
Scalable Link-Based Similarity Computation and Clustering

Xiaoxin Yin, Jiawei Han, and Philip S. Yu

Abstract Data objects in a relational database are cross-linked with each other via multi-typed links. Links contain rich semantic information that may indicate important relationships among objects, such as the similarities between objects. In this chapter we explore linkage-based clustering, in which the similarity between two objects is measured based on the similarities between the objects linked with them. We study a hierarchical structure called SimTree, which represents similarities in multi-granularity manner. This method avoids the high cost of computing and storing pairwise similarities but still thoroughly explore relationships among objects. We introduce an efficient algorithm for computing similarities utilizing the SimTree.

2.1 Introduction

As a process of partitioning data objects into groups according to their similarities with each other, clustering has been extensively studied for decades in different disciplines including statistics, pattern recognition, database, and data mining. There have been many clustering methods [1, 11, 15–17, 22], but most of them aim at grouping records in a *single table* into clusters using their own properties.

In many real applications, *linkages* among objects of different types can be the most explicit information available for clustering. For example, in a publication database (i.e., PubDB) in Fig. 2.1a, one may want to cluster each type of objects (authors, institutions, publications, proceedings, and conferences/journals), in order to find authors working on different topics, or groups of similar publications, etc. It is not so useful to cluster single type of objects (e.g., authors) based only on the properties of them, as those properties often provide little information relevant to the clustering task. On the other hand, the linkages between different types of objects (e.g., those between authors, papers, and conferences) indicate the relationships

X. Yin (✉)
Microsoft Research, Redmond, WA 98052, USA
e-mail: xyin@microsoft.com

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_2,

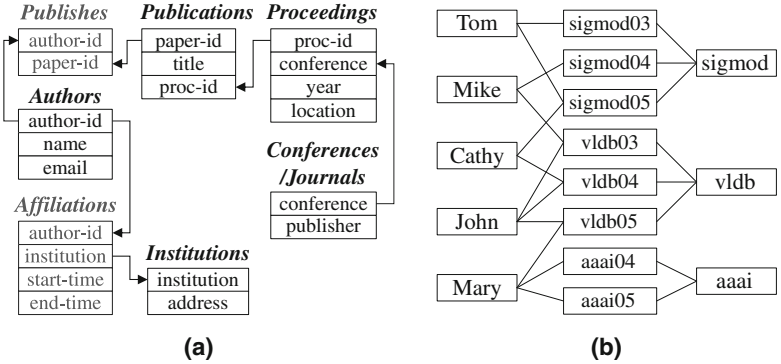


Fig. 2.1 A publication database (PubDB). (**a**) Database schema; (**b**) An example of linked objects

between objects and can help cluster them effectively. Such *linkage-based clustering* is appealing in many applications. For example, an online movie store may want to cluster movies, actors, directors, reviewers, and renters in order to improve its recommendation systems. In bioinformatics one may want to cluster genes, proteins, and their behaviors in order to discover their functions.

Clustering based on multi-typed linked objects has been studied in multi-relational clustering [14, 21], in which the objects of each type are clustered based on the objects of other types linked with them. Consider the mini-example in Fig. 2.1b. Authors can be clustered based on the conferences where they publish papers. However, such analysis is confined to direct links. For example, Tom publishes only SIGMOD papers, and John publishes only VLDB papers. Tom and John will have zero similarity based on direct links, although they may actually work on the same topic. Similarly, customers who have bought "*Matrix*" and those who have bought "*Matrix II*" may be considered dissimilar although they have similar interests.

The above example shows when clustering objects of one type, one needs to consider the similarities between objects of other types linked with them. For example, if it is known that SIGMOD and VLDB are similar, then SIGMOD authors and VLDB authors should be similar. Unfortunately, similarities between conferences may not be available, either. This problem can be solved by *SimRank* [13], in which the similarity between two objects is recursively defined as the average similarity between objects linked with them. For example, the similarity between two authors is the average similarity between the conferences in which they publish papers. In Fig. 2.1b "sigmod" and "vldb" have high similarity because they share many coauthors, and thus Tom and John become similar because they publish papers in similar conferences. In contrast, John and Mary do not have high similarity even they are both linked with "vldb05."

Although SimRank provides a good definition for similarities based on linkages, it is prohibitively expensive in computation. In [13] an iterative approach is proposed to compute the similarity between every pair of objects, which has quadratic complexity in both time and space, and is impractical for large databases.

Is it necessary to compute and maintain pairwise similarities between objects? Our answer is no for the following two reasons. First, hierarchy structures naturally exist among objects of many types, such as the taxonomy of animals and hierarchical categories of merchandise. Consider the example of clustering authors according to their research. There are groups of authors working on the same research topic (e.g., data integration or XML), who have high similarity with each other. Multiple such groups may form a larger group, such as the authors working on the same research area (e.g., database vs. AI), who may have weaker similarity than the former. As a similar example, the density of linkages between clusters of articles and words is shown in Fig. 2.2 (adapted from figure 5 (b) in [5]). We highlight four dense regions with dashed boxes, and in each dense region there are multiple smaller and denser regions. The large dense regions correspond to high-level clusters, and the smaller denser regions correspond to low-level clusters within the high-level clusters.

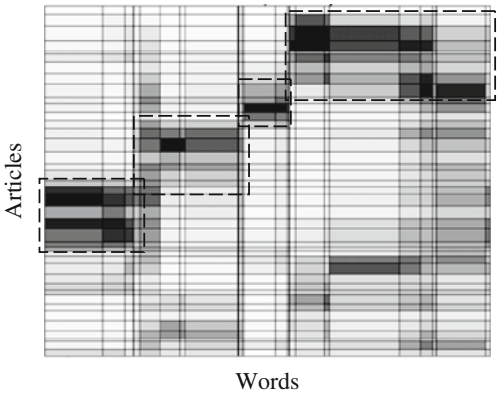


Fig. 2.2 Density of linkages between articles and words

Second, recent studies show that there exist power law distributions among the linkages in many domains, such as Internet topology and social networks [9]. Interestingly, based on our observation, such relationships also exist in the similarities between objects in interlinked environments. For example, Fig. 2.3 shows the distribution of pairwise SimRank similarity values between 4170 authors in DBLP database (the plot shows portion of values in each 0.005 range of similarity value). It can be seen that majority of similarity entries have very small values which lie within a small range (0.005 – 0.015). While only a small portion of similarity entries have significant values—1.4% of similarity entries (about 123 K of them) are greater than 0.1, and these values will play the major role in clustering. Therefore, we want to design a data structure that stores the significant similarity values and compresses those insignificant ones.

Based on the above two observations, we introduce a new hierarchical strategy to effectively prune the similarity space, which greatly speeds up the identification of similar objects. Taking advantage of the power law distribution of linkages, this strategy substantially reduces the number of pairwise similarities that need to be

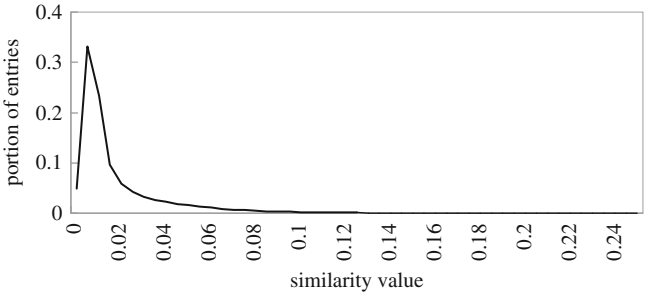


Fig. 2.3 Portions of similarity values

tracked, and the similarity between less similar objects will be approximated using aggregate measures.

We introduce a hierarchical data structure called SimTree as a compact representation of similarities between objects. Each leaf node of a SimTree corresponds to an object, and each non-leaf node contains a group of lower level nodes that are closely related to each other. SimTree stores similarities in a multi-granularity way by storing similarity between each two objects corresponding to sibling leaf nodes, and storing the overall similarity between each two sibling non-leaf nodes. Pairwise similarity is not pre-computed or maintained between objects that are not siblings. Their similarity, if needed, is derived based on the similarity information stored in the tree path. For example, consider the hierarchical categories of merchandise in Walmart. It is meaningful to compute the similarity between every two cameras, but not so meaningful to compute that for each camera and each TV, as an overall similarity between cameras and TVs should be sufficient.

Based on SimTree, we introduce LinkClus, an efficient and accurate approach for linkage-based clustering. At the beginning LinkClus builds a SimTree for each type of objects in a bottom-up manner, by finding groups of objects (or groups of lower level nodes) that are similar to each other. Because inter-object similarity is not available yet, the similarity between two nodes are measured based on the intersection size of their neighbor objects. Thus the initial SimTrees cannot fully catch the relationships between objects (e.g., some SIGMOD authors and VLDB authors have similarity 0).

LinkClus improves each SimTree with an iterative method, following the recursive rule that *two nodes are similar if they are linked with similar objects*. In each iteration it measures the similarity between two nodes in a SimTree by the average similarity between objects linked with them. For example, after one iteration SIGMOD and VLDB will become similar because they share many authors, which will then increase the similarities between SIGMOD authors and VLDB authors, and further increase that between SIGMOD and VLDB. We design an efficient algorithm for updating SimTrees, which merges the expensive similarity computations that go through the same paths in the SimTree. For a problem involving N objects and M linkages, LinkClus only takes $O(M(\log N)^2)$ time and $O(M+N)$ space (SimRank

takes $O(M^2)$ time and $O(N^2)$ space). Experiments on both real and synthetic data sets show that LinkClus achieves high accuracy and efficiency.

The rest of the chapter is organized as follows. We discuss related work in Section 2.2 and give an overview in Section 2.3. and 2.4 introduces SimTree, the hierarchical structure for representing similarities. The algorithms for building SimTrees and computing similarities are described in Section 2.5. Our performance study is reported in Section 2.6, and this study is concluded in Section 2.7.

2.2 Related Work

Clustering has been extensively studied for decades in different disciplines including statistics, pattern recognition, database, and data mining, with many approaches proposed [1, 11, 15–17, 22]. Most existing clustering approaches aim at grouping objects in a single table into clusters, using properties of each object. Some recent approaches [14, 21] extend previous clustering approaches to relational databases and measures similarity between objects based on the objects joinable with them in multiple relations.

In many real applications of clustering, objects of different types are given, together with linkages among them. As the attributes of objects often provide very limited information, traditional clustering approaches can hardly be applied, and linkage-based clustering is needed, which is based on the principle that two objects are similar if they are linked with similar objects.

This problem is related to bi-clustering [6] (or co-clustering [8], cross-association [5]), which aims at finding dense submatrices in the relationship matrix of two types of objects. A dense submatrix corresponds to two groups of objects of different types that are highly related to each other, such as a cluster of genes and a cluster of conditions that are highly related. Unlike bi-clustering that involves no similarity computation, LinkClus computes similarities between objects based on their linked objects. Moreover, LinkClus works on a more general problem as it can be applied to a relational database with arbitrary schema, instead of two types of linked objects. LinkClus also avoids the expensive matrix operations often used in bi-clustering approaches.

A bi-clustering approach [8] is extended in [4], which performs agglomerative and conglomerative clustering simultaneously on different types of objects. However, it is very expensive, —quadratic complexity for two types and cubic complexity for more types.

Jeh and Widom propose SimRank [13], a linkage-based approach for computing the similarity between objects, which is able to find the underlying similarities between objects through iterative computations. Unfortunately SimRank is very expensive as it has quadratic complexity in both time and space. The authors also discuss a pruning technique for approximating SimRank, which only computes the similarity between a small number of preselected object pairs. In the extended version of [13] the following heuristic is used: Only similarities between pairs

of objects that are linked with same objects are computed. With this heuristic, in Fig. 2.1b the similarity between SIGMOD and VLDB will never be computed. Neither will the similarity between Tom and John, Tom and Mike, etc. In general, it is very challenging to identify the right pairs of objects at the beginning, because many pairs of similar objects can only be identified after computing similarities between other objects. In fact this is the major reason that we adopt the recursive definition of similarity and use iterative methods.

A method is proposed in [10] to perform similarity searches by approximating SimRank similarities. It creates a large sample of random walk paths from each object and uses them to estimate the SimRank similarity between two objects when needed. It is suitable for answering similarity queries. However, very large samples of paths are needed for making accurate estimations for similarities. Thus it is very expensive in both time and space to use this approach for clustering a large number of objects, which requires computing similarities between numerous pairs of objects.

Wang et al. propose ReCom [20], an approach for clustering inter-linked objects of different types. ReCom first generates clusters using attributes and linked objects of each object, and then repeatedly refines the clusters using the clusters linked with each object. Compared with SimRank that explores pairwise similarities between objects, ReCom only explores the neighbor clusters and does not compute similarities between objects. Thus it is much more efficient but much less accurate than SimRank.

LinkClus is also related to hierarchical clustering [11, 17]. However, they are fundamentally different. Hierarchical clustering approaches use some similarity measures to put objects into hierarchies. While LinkClus uses hierarchical structures to represent similarities. This is related to the study in [3], which uses a tree structure to approximate metric spaces, although we do not require the objects to be in a metric space.

2.3 Overview

Linkage-based clustering is based on the principle that two objects are similar if they are linked with similar objects. For example, in a publication database (Fig. 2.1b), two authors are similar if they publish similar papers. The final goal of linkage-based clustering is to divide objects into clusters using such similarities. Figure 2.4 shows an example of three types of linked objects and clusters of similar objects which are inferred from the linkages. It is important to note that objects 12 and 18 do not share common neighbors, but they are linked to objects 22 and 24, which are similar because of their common linkages to 35, 37, and 38.

In order to capture the inter-object relationships as in the above example, we adopt the recursive definition of similarity in SimRank [13], in which the similarity between two objects x and y is defined as the average similarity between the objects linked with x and those linked with y.

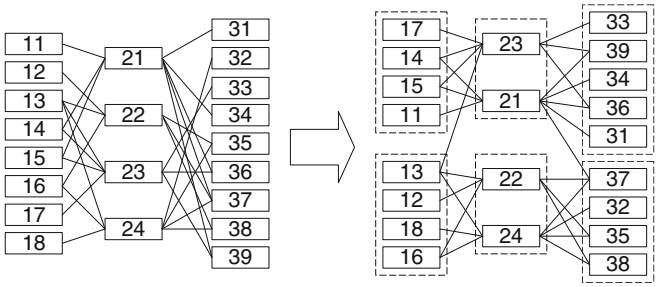


Fig. 2.4 Finding groups of similar objects

As mentioned in the introduction, a hierarchical structure can capture the hierarchical relationships among objects and can compress the majority of similarity values which are insignificant. Thus we use SimTree, a hierarchical structure for storing similarities in a multi-granularity way. It stores detailed similarities between closely related objects and overall similarities between object groups. We generalize the similarity measure in [13] to hierarchical environments and propose an efficient and scalable algorithm for computing similarities based on the hierarchical structure. Each node in a SimTree has at most c children, where c is a constant and is usually between 10 and 20. Given a database containing two types of objects, N objects of each type and M linkages between them, our algorithm takes $O(Nc+M)$ space and $O(M(\log_c N)^2c^2)$ time. This is affordable for very large databases.

2.4 SimTree: Hierarchical Representation of Similarities

In this section we describe SimTree, a new hierarchical structure for representing similarities between objects. Each leaf node of a SimTree represents an object (by storing its ID), and each non-leaf node has a set of child nodes, which are a group of closely related nodes of one level lower. An example SimTree is shown in Fig. 2.5a. The small gray circles represent leaf nodes, which must appear at the same level (which is level 0, the bottom level). The dashed circles represent non-leaf nodes. Each non-leaf node has at most c child nodes, where c is a small constant. Between each pair of sibling nodes n_i and n_j there is an undirected edge (n_i, n_j). (n_i, n_j) associated with a real value $s(n_i, n_j)$, which is the average similarity between all objects linked with n_i (or with its descendant objects if n_i is a non-leaf node) and those with n_j. $s(n_i, n_j)$ represents the overall similarity between the two groups of objects contained in n_i and n_j.

Another view of the same SimTree is shown in Fig. 2.5b, which better visualizes the hierarchical structure. The similarity between each pair of sibling leaf nodes is stored in the SimTree, while the similarity between two non-sibling leaf nodes is estimated using the similarity between their ancestor nodes. For example, suppose the similarity between n_7 and n_8 is needed, which is the average similarity between objects linked with n_7 and those with n_8. One can see that n_4 (or n_5) contains a

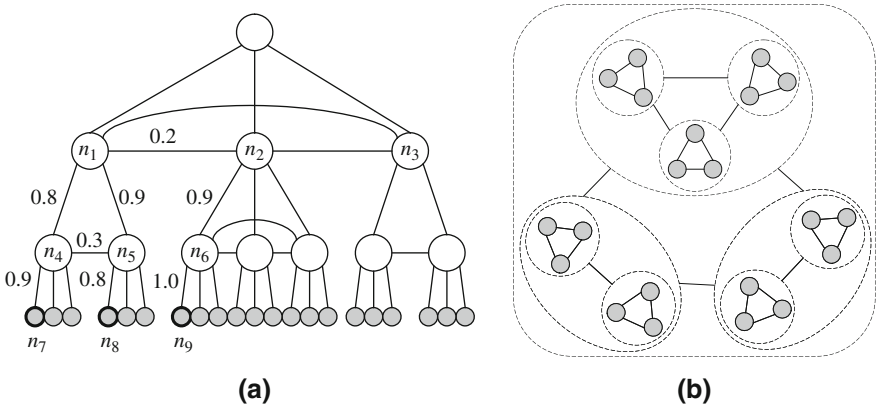


Fig. 2.5 An example SimTree. (**a**) Structure of a SimTree; (**b**) Another view of the SimTree

small group of leaf nodes including n_7 (or n_8), and we have computed $s(n_4, n_5)$ which is the average similarity between objects linked with these two groups of leaf nodes. Thus LinkClus uses $s(n_4, n_5)$ as the estimated similarity between n_7 and n_8. In a real application such as clustering products in Walmart, n_7 may correspond to a camera and n_8 to a TV. We can estimate their similarity using the overall similarity between cameras and TVs, which may correspond to n_4 and n_5, respectively. Similarly when the similarity between n_7 and n_9 is needed, LinkClus uses $s(n_1, n_2)$ as an estimation.

Such estimation is not always accurate, because a node may have different similarities to other nodes compared with its parent. LinkClus makes some adjustments to compensate for such differences, by associating a value to the edge between each node and its parent. For example, the edge (n_7, n_4) is associated with a real value $s(n_7, n_4)$, which is the ratio between (1) the average similarity between n_7 and all leaf nodes except n_4's descendants and (2) the average similarity between n_4 and those nodes. Similarly we can define $s(n_4, n_1)$, $s(n_6, n_2)$, etc. When estimating the similarity between n_7 and n_9, we use $s(n_1, n_2)$ as a basic estimation, use $s(n_4, n_1)$ to compensate for the difference between similarities involving n_4 and those involving n_1, and use $s(n_7, n_4)$ to compensate for n_7. The final estimation is $s(n_7, n_4) \cdot s(n_4, n_1) \cdot s(n_1, n_2) \cdot s(n_6, n_2) \cdot s(n_9, n_6) = 0.9 \cdot 0.8 \cdot 0.2 \cdot 0.9 \cdot 1.0 = 0.1296$.

In general, the similarity between two leaf nodes w.r.t. a SimTree is the product of the values of all edges on the path between them. Because this similarity is defined based on the path between two nodes, we call it *path-based similarity*.

Definition 1 (Path-based Node Similarity) Suppose two leaf nodes n_1 and n_k in a SimTree are connected by path $n_1 \to \cdots \to n_i \to n_{i+1} \to \cdots \to n_k$, in which n_i and n_{i+1} are siblings and all other edges are between nodes and their parents. The path-based similarity between n_1 and n_k is

$$sim_p(n_1, n_k) = \prod_{j=1}^{k-1} s(n_j, n_{j+1}). \tag{2.1}$$

Each node has similarity 1 with itself ($sim_p(n, n) = 1$).

Please note that within a path in Definition 1, there is only one edge that is between two sibling nodes, whose similarity is used as the basic estimation. The other edges are between parent and child nodes whose similarities are used for adjustments.

2.5 Building SimTrees

The inputs to LinkClus are objects of different types, with linkages between them. LinkClus maintains a SimTree for each type of objects to represent similarities between them. Each object is used as a leaf node in a SimTree. Figure 2.6 shows the leaf nodes created from objects of two types and the linkages between them.

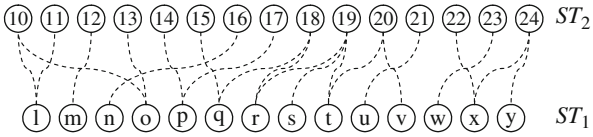


Fig. 2.6 Leaf nodes in two SimTrees

Initially each object has similarity 1 to itself and 0 to others. LinkClus first builds SimTrees using the initial similarities. These SimTrees may not fully catch the real similarities between objects, because inter-object similarities are not considered. LinkClus uses an iterative method to improve the SimTrees, following the principle that two objects are similar if and only if they are linked with similar objects. It repeatedly updates each SimTree using the following rule: The similarity between two nodes n_i and n_j is the average similarity between objects linked with n_i and those linked with n_j. The structure of each SimTree is also adjusted during each iteration by moving similar nodes together. In this way the similarities are refined in each iteration, and the relationships between objects can be discovered gradually.

2.5.1 Initializing SimTrees Using Frequent Pattern Mining

The first step of LinkClus is to initialize SimTrees using the linkages as shown in Fig. 2.6. Although no inter-object similarities are available at this time, the initial SimTrees should still be able to group related objects or nodes together, in order to provide a good base for further improvements.

Because only leaf nodes are available at the beginning, we initialize SimTrees from bottom level to top level. At each level, we need to efficiently find groups of

tightly related nodes and use each group as a node of the upper level. Consider a group of nodes $g = \{n_1, \ldots, n_k\}$. Let $neighbor(n_i)$ denote the set of objects linked with node n_i. Initially there are no inter-object similarities, and whether two nodes are similar depends on whether they are co-linked with many objects. Therefore, we define the tightness of group g as the number of objects that are linked with all group members, i.e., the size of intersection of $neighbor(n_1), \ldots, neighbor(n_k)$.

The problem of finding groups of nodes with high tightness can be reduced to the problem of finding frequent patterns [2]. A tight group is a set of nodes that are co-linked with many objects of other types, just like a frequent pattern is a set of items that co-appear in many transactions. Figure 2.7 shows an example which contains four nodes n_1, n_2, n_3, n_4 and objects linked with them. The nodes linked with each object are converted into a transaction, which is shown on the right side. It can be easily proved that the number of objects that are linked with all members of a group g is equal to the support of the pattern corresponding to g in the transactions. For example, nodes n_1 and n_2 are co-linked with two objects (#2 and #4), and pattern $\{n_1, n_2\}$ has support 2 (i.e., appear twice) in the transactions.

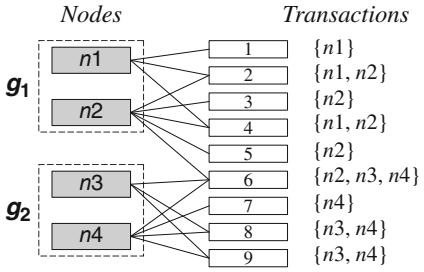


Fig. 2.7 Groups of tightly related nodes

Let $support(g)$ represent the number of objects linked with all nodes in g. When building a SimTree, we want to find groups with high support and at least min_size nodes. For two groups g and g' such that $g \subset g'$ and $support(g) = support(g')$, we prefer g'. Frequent pattern mining has been studied for a decade with many efficient algorithms. We can either find groups of nodes with support greater than a threshold using a frequent closed pattern mining approach [19], or find groups with highest support using a top-k frequent closed pattern mining approach [12]. LinkClus uses the approach in [19] which is very efficient on large data sets.

Now we describe the procedure of initializing a SimTree. Suppose we have built N_l nodes at level-l of the SimTree and want to build the nodes of level-$(l+1)$. Because each node can have at most c child nodes, and because we want to leave some space for further adjustment of the tree structure, we control the number of level-$(l+1)$ nodes to be between $\frac{N_l}{c}$ and $\frac{\alpha N_l}{c}$ $(1 < \alpha \leq 2)$. We first find groups of level-l nodes with sufficiently high support. Since there are usually many such groups, we select $\frac{\alpha N_l}{c}$ non-overlapping groups with high support in a greedy way, by repeatedly selecting the group with highest support that is not overlapped with previously selected groups. After selecting $\frac{\alpha N_l}{c}$ groups, we create a level-$(l+1)$

node based on each group. However, these groups usually cover only part of all level-l nodes. For each level-l node n_i that does not belong to any group, we want to put n_i into the group that is most connected with n_i. For each group g, we compute the number of objects that are linked with both n_i and some members of g, which is used to measure the connection between n_i and g. We assign n_i to the group with highest connection to n_i.

Figure 2.8 shows the two SimTrees built upon the leaf nodes in Fig. 2.6. The dashed lines indicate the leaf nodes in ST_2 that are linked with descendants of two non-leaf nodes n_a and n_b in ST_1. After building the initial SimTrees, LinkClus computes the similarity value associated with each edge in the SimTrees. As defined in Section 2.4, the similarity value of edge (n_a, n_b), $s(n_a, n_b)$, is the average similarity between objects linked with descendants of n_a and those of n_b. Because initially the similarity between any two different objects is 0, $s(n_a, n_b)$ can be easily computed based on the number of objects that are linked with both the descendants of n_a and those of n_b, without considering pairwise similarities. Similarly, the values associated with edges between child and parent nodes can also be computed easily.

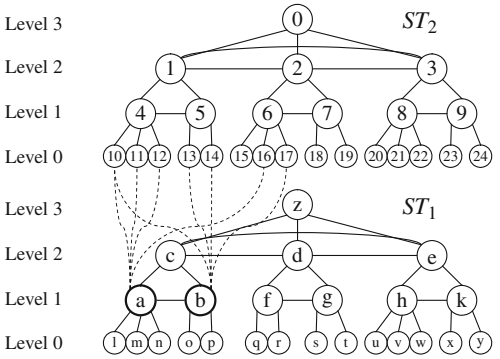


Fig. 2.8 Some linkages between two SimTrees

2.5.2 *Refining Similarity Between Nodes*

The initial SimTrees cannot fully catch the real similarities, because similarities between objects are not considered when building them. Therefore, LinkClus repeatedly updates the SimTrees, following the principle that the similarity between two nodes in a SimTree is the average similarity between the objects linked with them, which is indicated by other SimTrees. This is formally defined in this section.

We use $[n \sim n']$ to denote the linkage between two nodes n and n' in different SimTrees. We say there is a linkage between a non-leaf node n in ST_1 and a node n' in ST_2, if there are linkages between the descendant leaf nodes of n and the node n'.

Figure 2.8 shows the linkages between n_a, n_b, and leaf nodes in ST_2. In order to track the number of original linkages involved in similarity computation, we assign a weight to each linkage. By default the weight of each original linkage between two leaf nodes is 1. The weight of linkage $[n \sim n']$ is the total number of linkages between the descendant leaf nodes of n and n'.

In each iteration LinkClus updates the similarity between each pair of sibling nodes (e.g., n_a and n_b) in each SimTree, using the similarities between the objects linked with them in other SimTrees. The similarity between n_a and n_b is the average path-based similarity between the leaf nodes linked with n_a ($\{n_{10}, n_{11}, n_{12}, n_{16}\}$) and those with n_b ($\{n_{10}, n_{13}, n_{14}, n_{17}\}$). Because this similarity is based on linked objects, we call it *linkage-based similarity*. n_a (or n_b) may have multiple linkages to a leaf node n_i in ST_2, if more than one descendants of n_a are linked with n_i. Thus the leaf nodes in ST_2 linked with n_a are actually a multi-set, and the frequency of each leaf node n_i is $weight([n_a \sim n_i])$, which is the number of original linkages between n_a and n_i. The linkage-based similarity between n_a and n_b is defined as the average path-based similarity between these two multi-sets of leaf nodes, and in this way each original linkage plays an equal role.

Definition 2 (Linkage-Based Node Similarity) Suppose a SimTree ST is linked with SimTrees $ST_1, \ldots, ST_K$. For a node n in ST, *let* $NB_{ST_k}(n)$ denote the multi-set of leaf nodes in ST_k linked with n. Let $w_{n'n''}$ represent $weight([n' \sim n''])$. For two nodes n_a and n_b in ST, their linkage-based similarity $sim_l(n_a, n_b)$ is the average similarity between the multi-set of leaf nodes linked with n_a and that of n_b. We decompose the definition into several parts for clarity:
(The total weights between $NB_{ST_k}(n_a)$ and $NB_{ST_k}(n_b)$)

$$weight_{ST_k}(n_a, n_b) = \sum_{n \in NB_{ST_k}(n_a)} \sum_{n' \in NB_{ST_k}(n_b)} w_{n_a n} \cdot w_{n_b n'},$$

(The sum of weighted similarity between them)

$$sum_{ST_k}(n_a, n_b) = \sum_{n \in NB_{ST_k}(n_a)} \sum_{n' \in NB_{ST_k}(n_b)} w_{n_a n} \cdot w_{n_b n'} \cdot sim_p(n, n'),$$

(The linkage-based similarity between n_a and n_b w.r.t. ST_k)

$$sim_{ST_k}(n_a, n_b) = \frac{sum_{ST_k}(n_a, n_b)}{weight_{ST_k}(n_a, n_b)},$$

(The final definition of $sim_l(n_a, n_b)$)

$$sim_l(n_a, n_b) = \frac{1}{K} \sum_{k=1}^{K} sim_{ST_k}(n_a, n_b). \tag{2.2}$$

Equation (2.2) shows that if a SimTree ST is linked with multiple SimTrees, each linked SimTree plays an equal role in determining the similarity between nodes in ST. The user can also use different weights for different SimTrees according to the semantics.

2.5.3 Aggregation-Based Similarity Computation

The core part of LinkClus is how to iteratively update each SimTree by computing linkage-based similarities between different nodes. This is also the most computation-intensive part in linkage-based clustering. Definition 2 provides a brute-force method to compute linkage-based similarities. However, it is very expensive. Suppose each of two nodes n_a and n_b is linked with m leaf nodes. It takes $O(m^2 \log_c N)$ to compute $sim_l(n_a, n_b)$ ($\log_c N$ is the height of SimTree). Because some high-level nodes are linked with $\Theta(N)$ objects, this brute-force method requires $O(N^2 \log_c N)$ time, which is unaffordable for large databases.

Fortunately, we find that the computation of different path-based similarities can be merged if these paths are overlapped with each other.

Example 1 A simplified version of Fig. 2.8 is shown in Fig. 2.9, where $sim_l(n_a, n_b)$ is the average path-based similarity between each node in $\{n_{10}, n_{11}, n_{12}\}$ and each in $\{n_{13}, n_{14}\}$. For each node $n_k \in \{n_{10}, n_{11}, n_{12}\}$ and $n_l \in \{n_{13}, n_{14}\}$, their path-based similarity $sim_p(n_k, n_l) = s(n_k, n_4) \cdot s(n_4, n_5) \cdot s(n_5, n_l)$. All these six path-based similarities involve $s(n_4, n_5)$. Thus $sim_l(n_a, n_b)$, which is the average of them, can be written as

$$sim_l(n_a, n_b) = \frac{\sum_{k=10}^{12} s(n_k, n_4)}{3} \cdot s(n_4, n_5) \cdot \frac{\sum_{l=13}^{14} s(n_l, n_5)}{2}. \tag{2.3}$$

Equation (2.3) contains three parts: the average similarity between n_a and descendants of n_4, $s(n_4, n_5)$, and the average similarity between n_b and descendants of n_5. Therefore, we pre-compute the average similarity and total weights between n_a, n_b and n_4, n_5, as shown in Fig. 2.9. (The original linkages between leaf nodes do not affect similarities and thus have similarity 1.) We can compute $sim_l(n_a, n_b)$ using such aggregates, i.e., $sim_l(n_a, n_b) = \frac{0.9+1.0+0.8}{3} \times 0.2 \times \frac{0.9+1.0}{2} = 0.9 \times 0.2 \times 0.95 = 0.171$, and this is the average similarity between $3 \times 2 = 6$ pairs of leaf

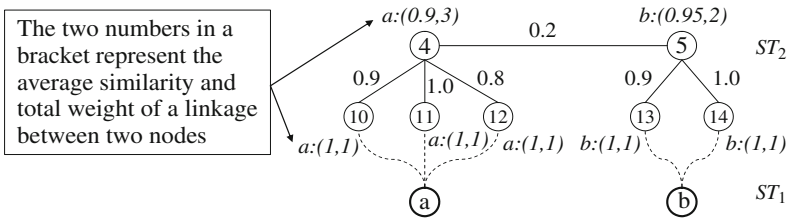


Fig. 2.9 Computing similarity between nodes

nodes. This is exactly the same as applying Definition 2 directly. But now we have avoided the pairwise similarity computation, since only the edges between siblings and parent–child are involved.

This mini-example shows the basic idea of computing linkage-based similarities. In a real problem n_a and n_b are often linked with many leaf nodes lying in many different branches of the SimTrees, which makes the computation much more complicated. The basic idea is still to merge computations that share common paths in SimTrees.

To facilitate our discussion, we introduce a simple data type called simweight, which is used to represent the similarity and weight associated with a linkage. A simweight is a pair of real numbers $\langle s, w \rangle$, in which s is the similarity of a linkage and w is its weight. We define two operations of simweight that are very useful in LinkClus.

Definition 3 (Operations of simweight) The operation of addition is used to combine two simweights corresponding to two linkages. The new similarity is the weighted average of their similarities, and the new weight is the sum of their weights:

$$\langle s_1, w_1 \rangle + \langle s_2, w_2 \rangle = \left\langle \frac{s_1 \cdot w_1 + s_2 \cdot w_2}{w_1 + w_2}, w_1 + w_2 \right\rangle. \tag{2.4}$$

The operation of multiplication is used to compute the weighted average similarity between two sets of leaf nodes. The new weight $w_1 \cdot w_2$ represents the number of pairs of leaf nodes between the two sets.

$$\langle s_1, w_1 \rangle \times \langle s_2, w_2 \rangle = \langle s_1 \cdot s_2, w_1 \cdot w_2 \rangle. \tag{2.5}$$

Lemma 1 *The laws of commutation, association, and distribution hold for the operations of simweight.*

LinkClus uses a simweight to represent the relationship between two nodes in different SimTrees. We use $NB_{ST}(n)$ to denote the multi-set of leaf nodes in ST linked with node n. For example, in Fig. 2.8 $NB_{ST_2}(n_a) = \{n_{10}, n_{11}, n_{12}, n_{16}\}$ and $NB_{ST_2}(n_b) = \{n_{10}, n_{13}, n_{14}, n_{17}\}$.

We first define the weight and similarity of a linkage between two non-leaf nodes in two SimTrees. A non-leaf node represents the set of its child nodes. Therefore, for a node n_a in ST_1 and a non-leaf node n_i in ST_2, the weight and similarity of linkage $[n_a \sim n_i]$ is the sum of weights and weighted average similarity between their child nodes. Furthermore, according to Definition 1, the similarity between two non-sibling nodes n_i and n_j on the same level of ST_2 can be calculated as

$$sim_p(n_i, n_j) = \\ s(n_i, parent(n_i)) \cdot sim_p(parent(n_i), parent(n_j)) \cdot s(n_j, parent(n_j)).$$

Thus we also incorporate $s(n_i, parent(n_i))$ (i.e., the ratio between average similarity involving n_i and that involving $parent(n_i)$) into the definition of linkage $[n_a \sim n_i]$. We use $sw_{n_a n_i}$ to denote the simweight of $[n_a \sim n_i]$.

Definition 4 Let n_a be a node in SimTree ST_1 and n_i be a non-leaf node in ST_2. Let $children(n_i)$ be all child nodes of n_i. The simweight of linkage $[n_a \sim n_i]$ is defined as

$$sw_{n_a n_i} = \sum_{\hat{n} \in children(n_i)} \langle s(\hat{n}, n_i), 1 \rangle \times sw_{n_a \hat{n}}. \tag{2.6}$$

(In (2.6) we use $\langle x, 1 \rangle \times \langle s, w \rangle$ as a convenient notation for $\langle x \cdot s, w \rangle$. Figures 2.9 and 2.10 shows $sw_{n_a n_i}$ and $sw_{n_b n_i}$ for each node n_i in ST_2.)

Using Definition 4, we formalize the idea in Example 1 as follows.

Lemma 2 *For two nodes n_a and n_b in SimTree ST_1, and two sibling non-leaf nodes n_i and n_j in ST_2, the average similarity and total weight between the descendant objects of n_i linked with n_a and those of n_j linked with n_b is*

$$sw_{n_a n_i} \times \langle s(n_i, n_j), 1 \rangle \times sw_{n_b n_j}.$$

(This corresponds to (2.3) if $i = 4$ and $j = 5$.)

We outline the procedure for computing the linkage-based similarity between n_a and n_b (see Fig. 2.10). $sim_l(n_a, n_b)$ is the average similarity between $NB_{ST_2}(n_a)$ and $NB_{ST_2}(n_b)$. We first compute the aggregated simweights $sw_{n_a n}$ and $sw_{n_b n}$ for each node n in ST_2, if n is an ancestor of any node in $NB_{ST_2}(n_a)$ or $NB_{ST_2}(n_b)$, as shown in Fig. 2.10. Consider each pair of sibling nodes n_i and n_j in ST_2 (e.g., n_4 and n_5), so that n_i is linked with n_a and n_j with n_b. According to Lemma 1, the average similarity and total weight between the descendant objects of n_i linked with n_a and those of n_j linked with n_b is $sw_{n_a n_i} \times \langle s(n_i, n_j), 1 \rangle \times sw_{n_b n_j}$. For example, $sw_{n_a n_4} = \langle 0.9, 3 \rangle$ (where the weight is 3 as n_a can reach n_4 via n_{10}, n_{11}, or n_{12}), $sw_{n_b n_5} = \langle 0.95, 2 \rangle$, and $s(n_4, n_5) = 0.2$. Thus $sw_{n_a n_4} \times \langle s(n_4, n_5), 1 \rangle \times$

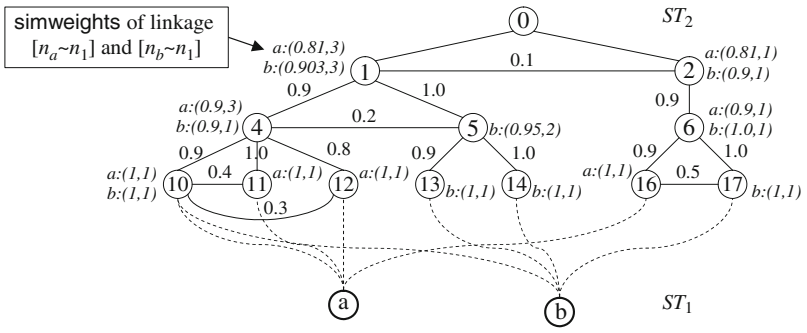


Fig. 2.10 Computing similarity between nodes

$sw_{n_b n_5} = \langle 0.171, 6\rangle$ (as in Example 1), which represents the average similarity and total weights between $\{n_{10}, n_{11}, n_{12}\}$ and $\{n_{13}, n_{14}\}$. We note that the weight is 6 as there are 6 paths between leaf nodes under n_4 linked with n_a and those under n_5 linked with n_b.

From the above example it can be seen that the effect of similarity between every pair leaf nodes in ST_2 will be captured when evaluating their ancestors that are siblings. For any two leaf nodes $\hat{n}_i$ and $\hat{n}_j$ in ST_2, there is only one ancestor of $\hat{n}_i$ and one of $\hat{n}_j$ that are siblings. Thus every pair of $\hat{n}_i, \hat{n}_j$ ($\hat{n}_i \in NB_{ST_2}(n_a)$, $\hat{n}_j \in NB_{ST_2}(n_b)$) is counted exactly once, and no redundant computation is performed. In general, $sim_l(n_a, n_b)$ can be computed using Theorem 1.

Theorem 1 (Sibling-Pair Based Similarity Computation) *Suppose n_a and n_b are two nodes in* SimTree *ST_1. Let $NB_{ST_2}(n_a)$ and $NB_{ST_2}(n_b)$ be the multi-sets of leaf nodes in* SimTree *ST_2 linked with n_a and n_b, respectively.*

$$\begin{aligned}&\langle sim_l(n_a, n_b), \text{ weight is ignored}\rangle = \\ &\sum_{n\in ST_2}\sum_{n_i,n_j\in children(n), n_i\neq n_j} sw_{n_a n_i} \times \langle s(n_i, n_j), 1\rangle \times sw_{n_b n_j} \\ &+ \sum_{n_i \in NB_{ST_2}(n_a)\bigcap NB_{ST_2}(n_b)} sw_{n_a n_i} \times sw_{n_b n_i}. \end{aligned} \tag{2.7}$$

The first term of (2.7) corresponds to similarities between different leaf nodes. For all leaf nodes under n_i linked with n_a and those under n_j linked with n_b, the effect of pairwise similarities between them is aggregated together as computed in the first term. The second term of (2.7) corresponds to the leaf nodes linked with both n_a and n_b. Only similarities between sibling nodes are used in (2.7), and thus we avoid the tedious pairwise similarity computation in Definition 2. In order to compute the linkage-based similarities between nodes in ST_1, it is sufficient to compute aggregated similarities and weights between nodes in ST_1 and nodes in other SimTrees. This is highly efficient in time and space as shown in Section 2.5.5.

Now we describe the procedure of computing $sim_l(n_a, n_b)$ based on Theorem 1.

Step 1: Attach the simweight of each original linkage involving descendants of n_a or n_b to the leaf nodes in ST_2.
Step 2: Visit all leaf nodes in ST_2 that are linked with both n_a and n_b to compute the second term in (2.7).
Step 3: Aggregate the simweights on the leaf nodes to those nodes on level-1. Then further aggregate simweights to nodes on level-2 and so on.
Step 4: For each node n_i in ST_2 linked with n_a and each sibling of n_i that is linked with n_b (we call it n_j), add $sw_{n_a n_i} \times \langle s(n_i, n_j), 1\rangle \times sw_{n_b n_j}$ to the first term of (2.7).

Suppose n_a is linked with m leaf nodes in ST_2 and n_b is linked with $O(m \cdot c)$ ones. It is easy to verify that the above procedure takes $O(mc \log_c N)$ time.

2.5.4 Iterative Adjustment of SimTrees

After building the initial SimTrees as described in Section 2.5.1, LinkClus needs to iteratively adjust both the similarities and structure of each SimTree. In Section 2.5.3 we have described how to compute the similarity between two nodes using similarities between their neighbor leaf nodes in other SimTrees. In this section we will introduce how to restructure a SimTree so that similar nodes are put together.

The structure of a SimTree is represented by the parent–child relationships, and such relationships may need to be modified in each iteration because of the modified similarities. In each iteration, for each node n, LinkClus computes n's linkage-based similarity with $parent(n)$ and the siblings of $parent(n)$. If n has higher similarity with a sibling node $\check{n}$ of $parent(n)$, then n will become a child of $\check{n}$, if $\check{n}$ has less than c children. The moves of low-level nodes can be considered as local adjustments on the relationships between objects and the moves of high-level nodes as global adjustments on the relationships between large groups of objects. Although each node can only be moved within a small range (i.e., its parent's siblings), with the effect of both local and global adjustments, the tree restructure is often changed significantly in an iteration.

The procedure for restructuring a SimTree is shown in Algorithm 1 (Fig. 2.11). LinkClus tries to move each node n to be the child of a parent node that is most similar to n. Because each non-leaf node $\check{n}$ can have at most c children, if there are more than c nodes that are most similar to $\check{n}$, only the top c of them can become children of $\check{n}$, and the remaining ones are reassigned to be children of other nodes similar to them.

After restructuring a SimTree ST, LinkClus needs to compute the value associated with every edge in ST. For each edge between two sibling nodes, their similarity is directly computed as in Section 2.5.3. For each edge between a node n and its parent, LinkClus needs to compute the average similarity between n and all leaf nodes except descendants of $parent(n)$ and that for $parent(n)$. It can be proved that the average linkage-based similarity between n and all leaf nodes in ST except descendants of a non-leaf node n' is

$$\frac{sum_{ST_k}(n, root(ST)) - sum_{ST_k}(n, n'))}{weight_{ST_k}(n, root(ST)) - weight_{ST_k}(n, n'))}. \tag{2.8}$$

Please note that (2.8) uses notations in Definition 2. With (2.8) we can compute the similarity ratio associated with each edge between a node and its parent. This finishes the computation of the restructured SimTree.

2.5.5 Complexity Analysis

In this section we analyze the time and space complexity of LinkClus. For simplicity, we assume there are two object types, each having N objects, and there are M linkages between them. Two SimTrees ST_1 and ST_2 are built for them. If there are

Algorithm 1 Restructure SimTree

Input: a SimTree ST to be restructured, which is linked with SimTrees $ST_1, \ldots, ST_k$.

Output: The restructured ST.

$S_c \leftarrow$ all nodes in ST except root *//S_c contains all child nodes*
$S_p \leftarrow$ all non-leaf nodes in ST *//S_p contains all parent nodes*
for each node n in S_c *//find most similar parent node for n*
  for each sibling node n' of $parent(n)$ (including $parent(n)$)
    compute $sim_l(n, n')$ using $ST_1, \ldots, ST_k$
  sort the set of $sim_l(n, n')$ for n
  $p^*(n) \leftarrow n'$ with maximum $sim_l(n, n')$
while($S_c \neq \emptyset$)
  for each node $\check{n} \in S_p$ *//assign children to $\check{n}$*
    $q^*(\check{n}) \leftarrow \{n | p^*(n) = \check{n}\}$
    if $|q^*(\check{n})| < c$
    then $children(\check{n}) \leftarrow q^*(\check{n})$
    else
      $children(\check{n}) \leftarrow c$ nodes in $q^*(\check{n})$ most similar to $\check{n}$
      $S_p \leftarrow S_p - \{\check{n}\}$
    $S_c \leftarrow S_c - children(\check{n})$
  for each node $n \in S_c$
    $p^*(n) \leftarrow n'$ with maximum $sim_l(n, n')$ and $n' \in S_p$
return ST

Fig. 2.11 Algorithm *Restructure* SimTree

more object types, the similarity computation between each pair of linked types can be done separately.

When a SimTree is built, LinkClus limits the number of nodes at each level. Suppose there are N_l nodes on level-l. The number of nodes on level-$(l+1)$ must be between $\frac{N_l}{c}$ and $\frac{\alpha N_l}{c}$ ($\alpha \in [1, 2]$ and usually $c \in [10, 20]$). Thus the height of a SimTree is $O(\log_c N)$.

In each iteration, LinkClus restructures each SimTree using similarities between nodes in the other SimTree and then updates the values associated with edges in each SimTree. When restructuring ST_1, for each node n in ST_1, LinkClus needs to compute its similarity to its parent and parent's siblings, which are at most c nodes. Suppose n is linked with m leaf nodes in ST_2. As shown in Section 2.5.3, it takes $O(mc \log_c N)$ time to compute the n's similarity with its parent or each of its parent's siblings. Thus it takes $O(mc^2 \log_c N)$ time to compute the similarities between n and these nodes.

There are N leaf nodes in ST_1, which have a total of M linkages to all leaf nodes in ST_2. In fact all nodes on each level in ST_1 have M linkages to all leaf nodes in ST_2, and there are $O(\log_c N)$ levels. Thus it takes $O(Mc^2(\log_c N)^2)$ time in total to compute the similarities between every node in ST_1 and its parent and parent's siblings.

In the above procedure, LinkClus processes nodes in ST_1 level by level. When processing the leaf nodes, only the simweights of linkages involving leaf nodes and nodes on level-1 of ST_1 are attached to nodes in ST_2. There are $O(M)$ such linkages, and the simweights on the leaf nodes in ST_2 require $O(M)$ space. In ST_2 LinkClus only compares the simweights of sibling nodes, thus it can also process the nodes level by level. Therefore, the above procedure can be done in $O(M)$ space. Each SimTree has $O(N)$ nodes, and it takes $O(c)$ space to store the similarity between each node and its siblings (and its parent's siblings). Thus the space requirement is $O(M + Nc)$.

It can be easily shown that the procedure for restructuring a SimTree (Algorithm 1) takes $O(Nc)$ space and $O(Nc \log c)$ time, which is much faster than computing similarities.

After restructuring SimTrees, LinkClus computes the similarities between each node and its siblings. This can be done using the same procedure as computing similarities between each node and its parent's siblings. Therefore, each iteration of LinkClus takes $O(Mc^2(\log_c N)^2)$ time and $O(M + Nc)$ space. This is affordable for very large databases.

2.6 Empirical Study

In this section we report experiments to examine the efficiency and effectiveness of LinkClus. LinkClus is compared with the following approaches: (1) *SimRank* [13], an approach that iteratively computes pairwise similarities between objects; (2) *ReCom* [20], an approach that iteratively clusters objects using the cluster labels of linked objects; (3) SimRank with fingerprints [10] (we call it *F-SimRank*), an approach that pre-computes a large sample of random paths from each object and uses the samples of two objects to estimate their SimRank similarity; (4) SimRank with pruning (we call it *P-SimRank*) [13], an approach that approximates SimRank by only computing similarities between pairs of objects reachable within a few links.

SimRank and F-SimRank are implemented strictly following their papers. (We use decay factor 0.8 for F-SimRank, which leads to highest accuracy in DBLP database.) ReCom is originally designed for handling web queries and contains a reinforcement clustering approach and a method for determining authoritativeness of objects. We only implement the reinforcement clustering method, because it may not be appropriate to consider authoritativeness in clustering. Since SimRank, F-SimRank, and P-SimRank only provide similarities between objects, we use CLARANS [16], a k-medoids clustering approach, for clustering using such similarities. CLARANS is also used in ReCom since no specific clustering method is discussed in [20]. We compare LinkClus using both hierarchical clustering and CLARANS.

All experiments are performed on an Intel PC with a 3.0 GHz P4 processor, 1GB memory, running Windows XP Professional. All approaches are implemented using

Visual Studio.Net (C#). In LinkClus, α is set to $\sqrt{2}$. We will discuss the influences of c (maximum number of children of each node) on accuracy and efficiency in the experiments.

2.6.1 Evaluation Measures

Validating clustering results is crucial for evaluating approaches. In our test databases there are predefined class labels for certain types of objects, which are consistent with our clustering goals. Jaccard coefficient [18] is a popular measure for evaluating clustering results, which is the number of pairs of objects in same cluster and with same class label, over that of pairs of objects either in same cluster or with same class label. Because an object in our databases may have multiple class labels but can only appear in one cluster, there may be many more pairs of objects with same class label than those in same cluster. Therefore we use a variant of Jaccard coefficient. We say two objects are correctly clustered if they share at least one common class label. The accuracy of clustering is defined as the number of object pairs that are correctly clustered over that of object pairs in same cluster. Higher accuracy tends to be achieved when number of clusters is larger. Thus we let each approach generate the same number of clusters.

2.6.2 DBLP Database

We first test on the DBLP database, which contains the following relations: (1) Authors, (2) Publications, which contains the publication title and the proceeding it is in, (3) Publish, which records which author publishes which publication, (4) Proceedings, which contains the proceeding title and which conferences it belongs to, and (5) Conferences. It is extracted from the XML data of DBLP [7]. We want to focus our analysis on the productive authors and well-known conferences,[1] and group them into clusters so that each cluster of authors (or conferences) are in a certain research area. We first select conferences that have been held for at least eight times. Then we remove conferences that are not about computer science or are not well known, and there are 154 conferences left. We select 4170 most productive authors in those conferences, each having at least 12 publications. The *Publications* relation contains all publications of the selected authors in the selected conferences. There are three types of objects to be clustered: 4170 authors, 2517 proceedings, and 154 conferences. Publications are not clustered because too limited information is known for them (about 65% of publications are associated with only one selected author).

[1] Here conferences refer to conferences, journals, and workshops. We are only interested in productive authors and well-known conferences because it is easier to determine the research fields related to each of them, from which the accuracy of clustering will be judged.

We manually label the areas of the most productive authors and conferences to measure clustering accuracy. The following 14 areas are considered: theory, AI, operating system, database, architecture, programming languages, graphics, networking, security, HCI, software engineering, information retrieval, bioinformatics, and CAD. For each conference, we study its historical call for papers to decide its area. Ninety percent of conferences are associated with a single area. The other 10% are associated with multiple areas, such as KDD (database and AI). We analyze the research areas of 400 most productive authors. For each of them, we find her home page and infer her research areas from her research interests. If no research interests are specified, we infer her research areas from her publications. On average each author is interested in 2.15 areas. In the experiments each type of objects are grouped into 20 clusters, and the accuracy is tested based on the class labels.

We perform 20 iterations for SimRank, P-SimRank, ReCom, and LinkClus[2] (not including the initialization process of each approach). In F-SimRank we draw a sample of 100 paths (as in [10]) of length 20 for each object, so that F-SimRank can use comparable information as SimRank with 20 iterations. The accuracies of clustering authors and conferences of each approach are shown in Fig. 2.12 (a) and (b), in which the x-axis is the index of iterations.

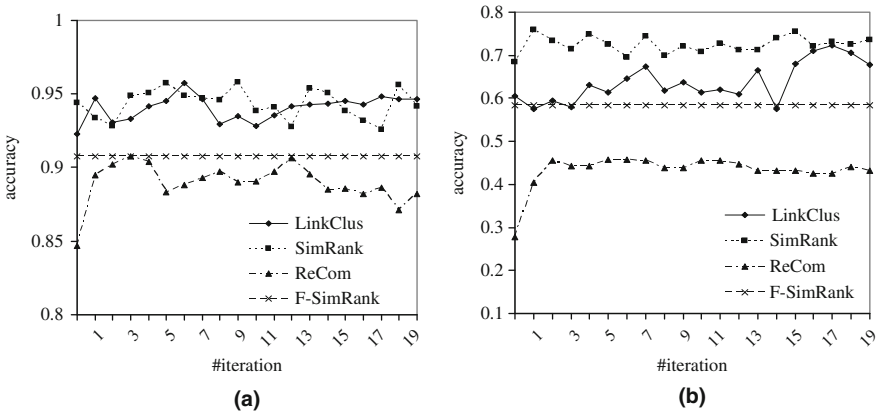


Fig. 2.12 Accuracy on DBLP. (**a**) DBLP.Authors; (**b**) DBLP.Conferences

From Fig. 2.12 one can see that SimRank is most accurate, and LinkClus achieves similar accuracy as SimRank. The accuracies of ReCom and F-SimRank are significantly lower. The error rates (i.e., 1 − accuracy) of ReCom and F-SimRank are about twice those of SimRank and LinkClus on authors, and 1.5 times those of them on conferences. One interesting observation is that more

[2] Since no frequent patterns of conferences can be found using the proceedings linked to them, LinkClus uses authors linked with conferences to find frequent patterns of conferences, in order to build the initial SimTree for conferences.

iterations do not necessarily lead to higher accuracy. This is probably because cluster labels are not 100% coherent with data. In fact this is common for iterative clustering algorithms.

In the above experiment, LinkClus generates 20 clusters directly from the SimTrees: Given a SimTree, it first finds the level in which the number of nodes is most close to 20. Then it either keeps merging the most similar nodes if the number of nodes is more than 20, or keeps splitting the node with most descendant objects if otherwise, until 20 nodes are created. We also test LinkClus using CLARANS with the similarities indicated by SimTrees. The maximum accuracies and running time of different approaches are shown in Table 2.1. (The running time per iteration of F-SimRank is its total running time divided by 20.) One can see that the accuracy of LinkClus with CLARANS is slightly higher than that of LinkClusand is close to that of SimRank. While SimRank is much more time consuming than other approaches.

Table 2.1 Performances on DBLP without keywords

	Maximum accuracy		Time/iteration (s)
	Authors	Conferences	
LinkClus	0.9574	0.7229	76.74
LinkClus-Clarans	0.9529	0.7523	107.7
SimRank	0.9583	0.7603	1020
ReCom	0.9073	0.4567	43.1
F-SimRank	0.9076	0.5829	83.6

In many methods of linkage-based clustering there is a trade-off between accuracy and efficiency. This trade-off is shown in Fig. 2.13, which contains the "accuracy vs. time" plots of SimRank, ReCom, LinkClus with different c's (8–22, including $c = 16$ with CLARANS), and F-SimRank with sample size of 50, 100,

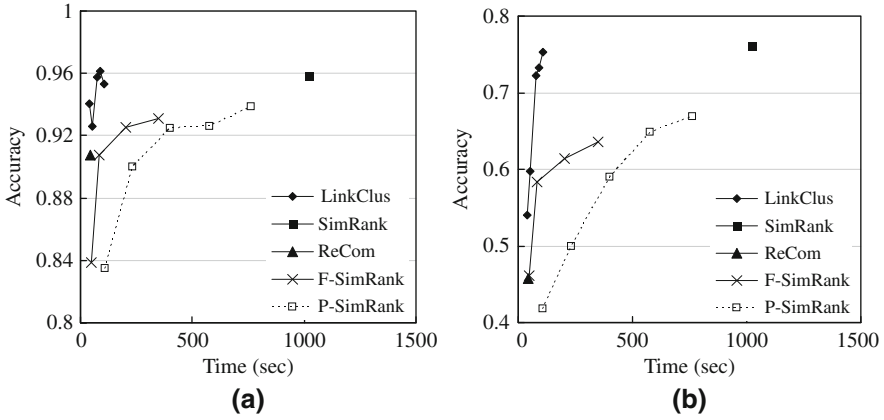


Fig. 2.13 Accuracy vs. time on DBLP w/o keywords. (**a**) DBLP.Authors; (**b**) DBLP.Conferences

200, and 400. It also includes SimRank with pruning (P-SimRank), which uses the following pruning method: For each object x, we only compute its similarity with the top-k objects that share most common neighbors with x within two links (k varies from 100 to 500). In these two plots, the approaches in the top-left region are good ones as they have high accuracy and low running time. It can be clearly seen that LinkClus greatly outperforms the other approaches, often in both accuracy and efficiency. In comparison, pruning technique of SimRank does not improve much on efficiency, because it requires using hashtables to store similarities, and an access to a hashtable is 5–10 times slower than that to a simple array.

2.6.3 Synthetic Databases

In this section we test the scalability and accuracy of each approach on synthetic databases. Figure 2.14 shows an example schema of a synthetic database, in which R_1, R_2, R_3, R_4 contain objects, and R_5, R_6, R_7, R_8 contain linkages. We use $RxTyCzSw$ to represent a database with x relations of objects, each having y objects which are divided into z clusters, and each object has w linkages to objects of another type (i.e., selectivity is w). In each relation of objects R_i, the x objects are randomly divided into z clusters. Each cluster is associated with two clusters in each relation of objects linked with R_i. When generating linkages between two linked relations R_i and $R_{i\%4+1}$, we repeat the following procedure for $x \cdot w$ times: Randomly select an object o in R_i and find the two clusters in $R_{i\%4+1}$ associated with the cluster of o. Then generate a linkage between o and a randomly selected object in these two clusters with probability $(1 - noise\_ratio)$ and generate a linkage between o and a randomly selected object with probability $noise\_ratio$. The default value of $noise\_ratio$ is 20%. It is shown in previous experiments that in most cases each approach can achieve almost the highest accuracy in 10 iterations, we use 10 iterations in this section. We let each approach generate z clusters for a database $RxTyCzSw$. For LinkClus we use $c = 16$ and do not use CLARANS.

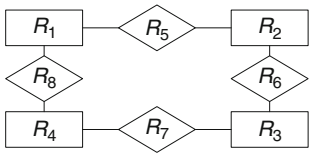


Fig. 2.14 The schema of a synthetic database

We first test scalability w.r.t. the number of objects. We generate databases with 5 relations of objects, 40 clusters in each of them, and selectivity 10. The number of objects in each relation varies from 1000 to 5000. The running time and accuracy of each approach are shown in Fig. 2.15. The time/iteration of F-SimRank is the total time divided by 10. With other factors fixed, theoretically the running time of LinkClus is $O(N(\log N)^2)$, that of $SimRank$ is $O(N^2)$, and those of ReCom and

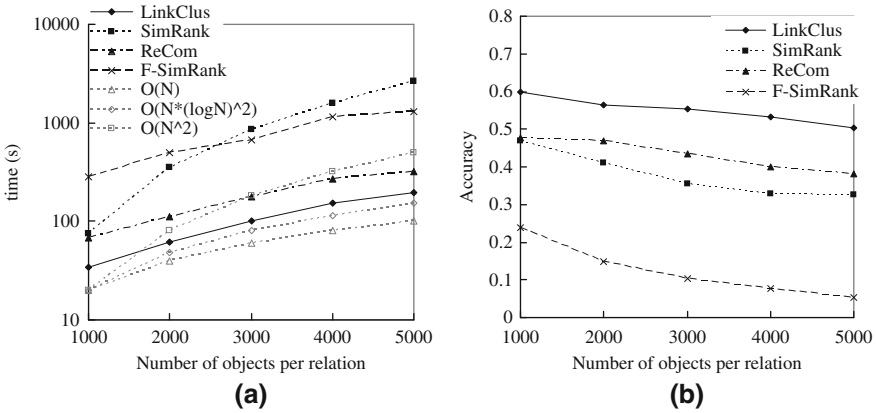


Fig. 2.15 Performances on R5T*C40S10. (**a**) Time/iteration; (**b**) Accuracy

F-SimRank are $O(N)$. We also show the trends of these bounds and one can see that the running time of the approaches are consistent with theoretical derivations. LinkClus achieves highest accuracy, followed by ReCom and then SimRank, and F-SimRank is least accurate. The possible reason for LinkClus and ReCom achieving high accuracy is that they group similar objects into clusters (or tree nodes) in the clustering process. Because clusters are clearly generated in data, using object groups in iterative clustering is likely to be beneficial.

In the last experiment the accuracy of each approach keeps decreasing as the number of objects increases. This is because the density of linkages decreases as cluster size increases. In R5T1000C40S10, each cluster has only 25 objects, each having 10 linkages to the two related clusters (50 objects) in other relations. In R5T5000C40S10, each cluster has 125 objects and the two related clusters have 250 objects, which makes the linkages much sparser. In the second experiment we increase the number of objects and clusters together to keep density of linkages fixed. Each cluster has 100 objects, and the number of objects per relation varies from 500 to 20000. In the largest database there are 100 K objects and 1 M linkages. The running time and accuracy of each approach are shown in Fig. 2.16.[3] ReCom and F-SimRank are unscalable as their running time is proportional to the number of objects times the number of clusters, because they compute similarities between each object and each cluster medoid. The accuracies of LinkClus and SimRank do not change significantly, even the numbers of objects and clusters grow 40 times.

Then we test each approach on databases with different selectivities, as shown in Fig. 2.17. We generate databases with five relations of objects, each having 4000 objects and 40 clusters. The selectivity varies from 5 to 25. The running time of LinkClus grows linearly and that of SimRank quadratically with the selectivity, and

[3] We do not test SimRank and F-SimRank on large databases because they consume too much memory.

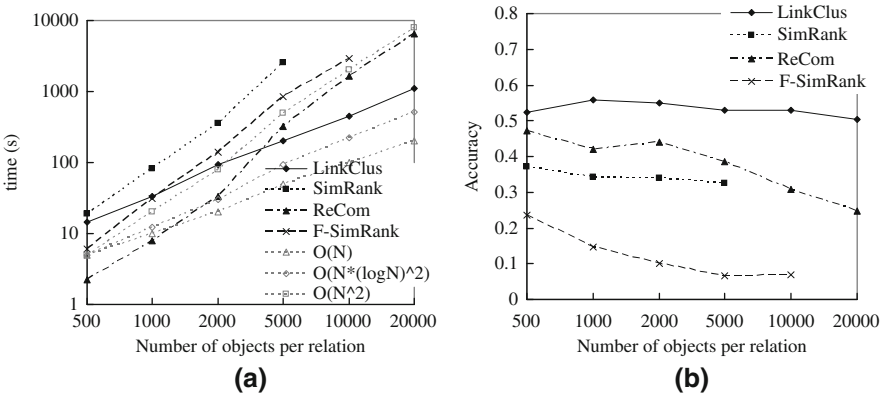


Fig. 2.16 Performances on R5T*C*S10. (**a**) Time/iteration; (**b**) Accuracy

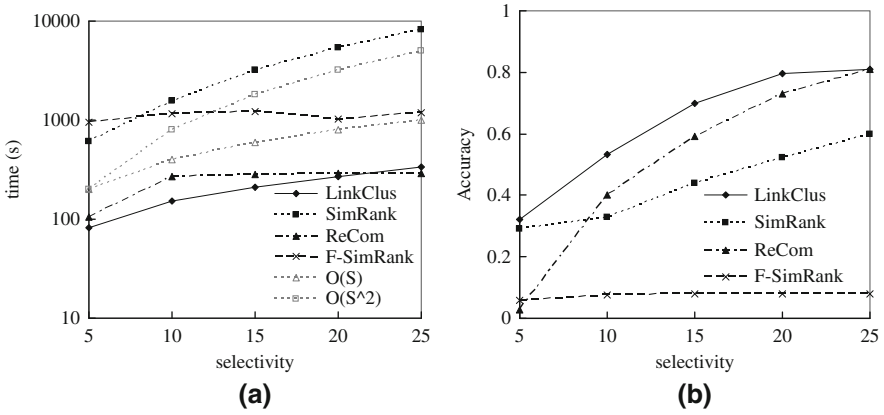


Fig. 2.17 Performances on R5T4000C40S*. (**a**) Time/iteration; (**b**) Accuracy

those of ReCom and F-SimRank are only slightly affected. These are consistent with theoretical derivations. The accuracies of LinkClus, SimRank, and ReCom increase quickly when selectivity increases, showing that density of linkages is crucial for accuracy. The accuracy of F-SimRank remains stable because it does not use more information when there are more linkages.

Finally, we test the accuracy of each approach on databases with different noise ratios, as shown in Fig. 2.18. We change noise ratio from 0 to 0.4. The accuracies of LinkClus, SimRank, and F-SimRank decrease with a stable rate when noise ratio increases. ReCom is most accurate when noise ratio is less than 0.2, but is least accurate when noise ratio is greater than 0.2. It shows that LinkClus and SimRank are more robust than ReCom in noisy environments.

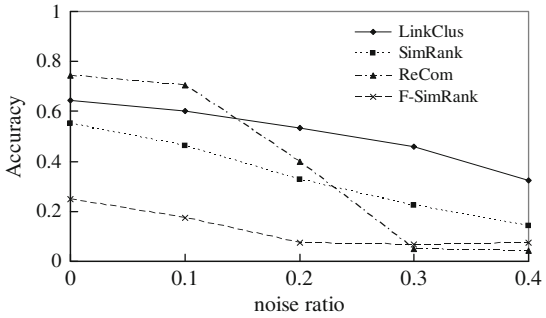


Fig. 2.18 Accuracy vs. noise ratio on R5T4000C40S10

2.7 Conclusions

In this chapter we propose a highly effective and efficient approach of linkage-based clustering, LinkClus, which explores the similarities between objects based on the similarities between objects linked with them. We propose similarity-based hierarchical structure called SimTree as a compact representation for similarities, and propose an efficient algorithm for computing similarities, which avoiding pairwise computations by merging similarity computations that go through common paths. Experiments show LinkClus achieves high efficiency, scalability, and accuracy in clustering multi-typed linked objects.

References

1. C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *SIGMOD*, Philadelphia, PA, 1999.
2. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, Washington, DC, 1993.
3. Y. Bartal. On approximating arbitrary metrics by tree metrics. In *STOC*, Dallas, TX, 1998.
4. R. Bekkerman, R. El-Yaniv, and A. McCallum. Multi-way distributional clustering via pairwise interactions. In *ICML*, Bonn, Germany, 2005.
5. D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD*, Seattle, WA, 2004.
6. Y. Cheng and G. M. Church. Biclustering of expression data. In *ISMB*, La Jolla, CA, 2000.
7. DBLP Bibliography. www.informatik.uni-trier.de/~ley/db/
8. I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *KDD*, Washington, DC, 2003.
9. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *SIGCOMM*, Cambridge, MA, 1999.
10. D. Fogaras and B. Rácz. Scaling link-base similarity search. In *WWW*, Chiba, Japan, 2005.
11. S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *SIGMOD*, Seattle, WA, 1998.
12. J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *ICDM*, Maebashi City, Japan, 2002.

13. G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. In *KDD*, Edmonton, Canada, 2002.
14. M. Kirsten and S. Wrobel. Relational distance-based clustering. In *ILP*, Madison, WI, 1998.
15. J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium*, Berkeley, CA, 1967.
16. R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *VLDB*, Santiago de Chile, Chile, 1994.
17. R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.
18. P.-N. Tan, M. Steinbach, and W. Kumar. *Introdution to data mining*. Addison-Wesley, New York, NY 2005.
19. J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *KDD*, Washington, DC, 2003.
20. J. D. Wang, H. J. Zeng, Z. Chen, H. J. Lu, L. Tao, and W. Y. Ma. ReCoM: Reinforcement clustering of multi-type interrelated data objects. In *SIGIR*, Toronto, Canada, 2003.
21. X. Yin, J. Han, and P. S. Yu. Cross-relational clustering with user's guidance. In *KDD*, Chicago, IL, 2005.
22. T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *SIGMOD*, Montreal, Canada, 1996.

Chapter 3
Community Evolution and Change Point Detection in Time-Evolving Graphs

Jimeng Sun, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos

Abstract How can we find communities in dynamic networks of social interactions, such as who calls whom, who emails whom, or who sells to whom? How do we store a large volume of IP network source–destination connection graphs, which grow over time? In this chapter, we study these two fundamental problems on time-evolving graphs and exploit the subtle connection between pattern mining and compression. We propose a pattern mining method, *GraphScope*, that automatically reveals the underlying communities in the graphs, as well as the change points in time. Our method needs no human intervention, and it is carefully designed to operate in a streaming fashion. Moreover, it is based on lossless compression principles. Therefore, in addition to revealing the fundamental structure of the graphs, the discovered patterns naturally lead to an excellent storage scheme for graph streams. Thus, our proposed *GraphScope* method unifies and solves both the mining and the compression problem (1) by producing meaningful time-evolving patterns agreeing with human intuition and (2) by identifying key change points in several real large time-evolving graphs. We demonstrate its efficiency and effectiveness on real data sets from several domains.

3.1 Introduction

Graphs and networks arise naturally in a wide range of disciplines and application domains, since they capture the general notion of an association between two entities. However, the aspect of time has only recently begun to receive some attention [19, 26]. Some examples of the time-evolving graphs include the following: (a) Network packets indicate ongoing communication between source and destination hosts like the `NETWORK` data set in our experiment; (b) Email

J. Sun (✉)
IBM TJ Watson Research Center, Hawthorne, NY, USA
e-mail: jimengsun@gmail.com

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_3,

networks associate a sender and a recipient at a given date, like the ENRON data set (http://www.cs.cmu.edu/enron/) that we use in the experiment; (c) Call detail records in telecommunications networks associate a caller with a callee. The set of all conversation pairs over each week forms a graph that evolves over time, like the publicly available "CELLPHONE" data set of MIT users calling each other (http://reality.media.mit.edu/download.php); (d) Transaction data: in a financial institution, who accessed what account, and when; (e) In a database compliance setting [2], again we need to record which user accessed what data item and when; and (f) Market-basket transaction data, which associate customers with products purchased at one visit to the store.

To complicate matters further, large amounts of data such as those in the above examples are continuously collected. Therefore, batch methods for pattern discovery are not sufficient. Additionally, the volume of the data poses significant challenges on storing such data. In summary, there are two key problems that need to be addressed:

(P1) *Mining*: Which groups or communities of nodes are associated to each other and how do these relationships evolve over time? Moreover, we want to answer these questions (a) *without* requiring any user-defined parameters, and (b) in a *stream* fashion.
(P2) *Compression*: How can such dynamically evolving streams of pairwise relationships be efficiently stored, without any loss of information?

For example, we want to answer questions such as: How do the network hosts interact with each other? What kind of host groups are there, e.g., inactive/active hosts; servers; scanners? Who emails whom? Do the email communities in a organization such as ENRON remain stable, or do they change between workdays (e.g., business-related) and weekends (e.g., friend and relatives), or during major events (e.g., the FBI investigation and the CEO resignation)? Which types of customers buy which kinds of products? Are there seasonal patterns in these relationships (e.g., winter and summer, or Thanksgiving and Christmas)? Additionally, we want to efficiently store data of such interactions, losslessly.

We propose GraphScope, which addresses both of the above problems simultaneously. More specifically, GraphScope is an efficient, adaptive compression scheme on time-evolving graphs. Unlike many existing techniques, it requires no user-defined parameters, and it operates completely automatically, based on the Minimum Description Length (MDL) principle. Furthermore, it adapts to the dynamic environment by automatically finding the communities and determining good *change-points* in time.

In this chapter we consider bipartite graphs, which treat source and destination nodes separately (see example in Fig. 3.2). As will become clear later, we discover separate source and destination partitions, which are desirable in several application domains. Nonetheless, our methods can be easily modified to deal with unipartite graphs, by constraining the source partitions to be the same with the destination partitions, as was done in [5].

The main insight of dealing with such graphs is to group "similar" sources together into *source groups* (or *row groups*), and also "similar" destinations together, into *destination groups* (or *column groups*). Figure 3.3 shows how much more orderly (and easier to compress) the adjacency matrix of a graph is, after we strategically re-order its rows and columns. The exact definition of "similar" is actually simple, and rigorous: the most similar source partitions for a given source node is the one that leads to best compression. See Section 3.4 for more details.

Furthermore, if these communities (source- and destination partitions) do not change much over time, consecutive snapshots of the evolving graphs have similar descriptions and can also be grouped together into a time segment, to achieve better compression. Whenever a new graph snapshot cannot fit well into the old segment (in terms of compression), GraphScope introduces a change point, and starts a new segment at that timestamp. Those change points often detect drastic discontinuities in time. For example on the ENRON data set, the change points all coincide with important events related to the ENRON company, as shown in Fig. 3.1 (more details in Section 3.6.2).

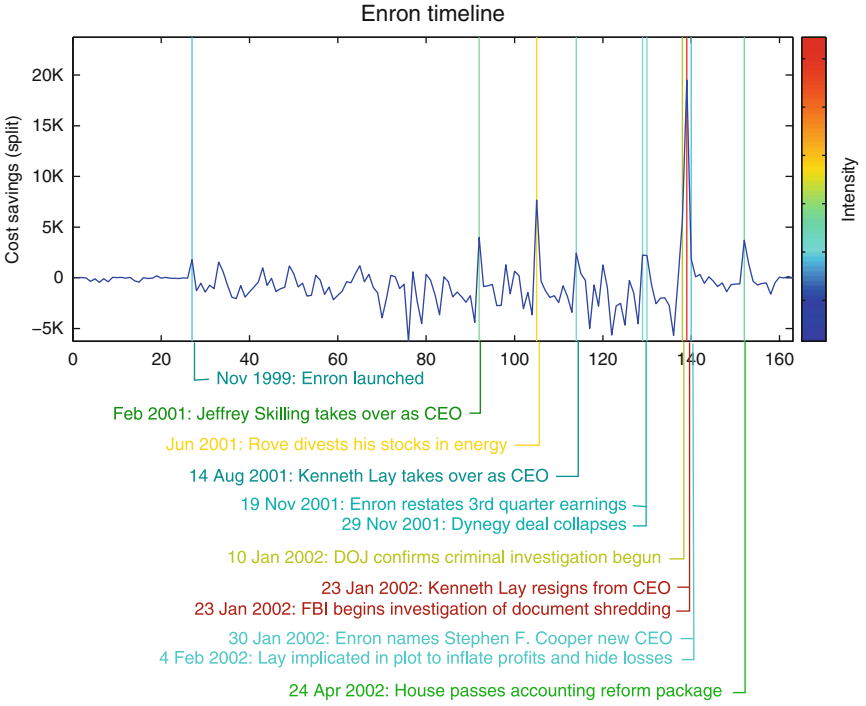


Fig. 3.1 ENRON data set (best viewed in color). Relative compression cost versus time. Large cost indicates change points, which coincide with the key events; E.g., at time-tick 140 (Feb 2002), CEO Ken Lay was implicated in fraud

Contributions: Our proposed approach, GraphScope, provides a unified treatment of the two fundamental problems of mining and compression of evolving graphs, and it has the following key properties:

- *Adaptivity*: It can effectively track communities over time, discovering both communities as well as change points in time, that agree with human intuition.
- *Streaming ability*: It is fast, incremental and scalable for the streaming environment.
- *Space efficiency*: It provides a lossless storage scheme which achieves very high compression ratios (20:1), on all the real data sets in our experiments.
- *Parameter-free*: This is the major point of difference with all other community-tracking methods: GraphScope is completely automatic, requiring no parameters from the user (like number of communities, thresholds to assess community drifts). Instead, it is based on sound information theory principles, specifically, the MDL idea.

We demonstrate the efficiency and effectiveness of our approach in both compressing evolving graphs as well as discovering and tracking the key patterns in the real data from several domains.

The rest of the chapter is organized as follows: Section 3.2 reviews the related work. Section 3.3 introduces some necessary definitions and formalizes the problem. Section 3.4 presents the compression objective function. Section 3.5 presents our proposed method for finding optimal solution, Section 3.6 shows the experimental evaluation and Section 3.7 concludes.

3.2 Related Work

Here we discuss related work from three areas: mining static graphs, mining dynamic graphs, and stream mining.

3.2.1 Mining Static Graphs

Graph mining has been a very active area in data mining community. From the exploratory aspect, Faloutsos et al. [11] have shown the power law distribution on the Internet graph. Kumar et al. [18] discovered the bow tie model for web graphs.

From the algorithmic aspect, graph partitioning has attracted much interest, with prevailing methods being METIS [16] and spectral partitioning [23]. Even in these top-performing methods, users must specify the number of partitions k. Moreover, they typically also require a measure of imbalance between the two pieces of each cut.

Information-theoretic co-clustering (ITCC) [9] performs simultaneously clustering rows and columns of a normalized contingency table or a two-dimensional probability distribution, where the number of clusters have to be specified. The cross-association method (CA) [6] formulates the co-clustering problem as a binary matrix compression problem.

Since common representation of a graph is sparse matrix (adjacency list), the sparse iterative methods such as Lanczos algorithm are especially relevant [14]. Column selection methods [10, 17] provide an alternative way of summarizing graphs. Namely, they choose a subset of columns as bases and summarize the rest columns as linear combinations of the selective columns.

All these methods deal with static matrices or graphs, while GraphScope is designed to work with dynamic streams. Moreover, most of methods except for cross-association require some user-defined parameters, which may be difficult to set and which may dramatically affect the final result as observed in [17].

3.2.2 Mining Dynamic Graphs

From the exploratory viewpoint, Leskovec et al. [19] discover the shrinking diameter phenomena on time-evolving graphs. Backstrom et al. [4] study community evolution in social networks.

From the algorithmic aspect, Sun et al. [26] present dynamic tensor analysis which incrementally summarizes tensor streams (high-order graph streams) as smaller core tensor streams and projection matrices. This method still requires user-defined parameters (like the size of the core tensor). Moreover, it gives lossy compression. Aggarwal and Yu [1] propose a method (1) to selectively store a subset of graphs to approximate the entire graph stream and (2) to find community changes in time-evolving graphs based on the user specified time interval and the number of communities. Lin et al. [20, 21] provide a series of dynamic soft clustering techniques that allow nodes to belong to multiple clusters with different probability.

Again, our GraphScope avoids all these user-defined parameters.

3.2.3 Stream Mining

Data streams have been extensively studied in recent years. The goal is to process the incoming data efficiently without recomputing from scratch and without buffering much historical data. The two surveys [3, 22] discuss many data streams algorithms. Among them, "Sketches" is a powerful technique that uses a compact structure to estimate many important statistics, such as the Lp-norm [7, 15] of an unbounded stream. Garofalakis and Gibbons [13] proposed single-pass algorithms for approximating the largest wavelet coefficients using "Sketches."

Ganti et al. [12] propose a generic framework for stream mining. For multiple streams, statStream [28] uses the DFT to summarize streams within a finite window and then compute the highest pairwise correlations among all pairs of streams, at each timestamp. SPIRIT [24] applies incremental SVD to summarize multiple streams into a small number of hidden variables.

All the stream mining works deal with time-series type of streams, while we focus on graph streams.

3.3 Problem Definition

In this section, we formally introduce the notation and formulate the problems.

3.3.1 Notation and Definition

Let's start with some definitions and naming conventions. Calligraphic letters always denote *graph streams* or *graph stream segments* (consisting of one or more graph snapshots), while individual graph snapshots are denoted by non-calligraphic, upper case letters. Superscripts in parentheses denote either timestamps t or graph segment indices s, accordingly. Similarly, subscripts denote either individual nodes i, j or node partitions p, q. All notations are described in Table 3.1.

Table 3.1 Definitions of symbols

Sym.	Definition
$\mathcal{G}, \mathcal{G}^{(s)}$	Graph stream, Graph segment
t	Timestamp, $t \geq 1$
m, n	Number of source (destination) nodes
$G^{(t)}$	Graph at time t ($m \times n$ adjacency matrix)
i, j	Node indices, $1 \leq i \leq m$, $1 \leq j \leq n$
$G^{(t)}_{i,j}$	Indicator for edge (i, j) at time t
s	Graph segment index, $s \geq 1$.
t_s	Starting time of sth segment
k_s,ℓ_s	Number of source (dest.) partitions for segment s
p, q	Partition indices, $1 \leq p \leq k_s$, $1 \leq q \leq \ell_s$
$I^{(s)}_p$	Set of sources belonging to the pth partition, during the sth segment
$J^{(s)}_q$	Similar to $I^{(s)}_p$, but for destination nodes
$m^{(s)}_p$	Source partition size, $m^{(s)}_p \equiv \lvert I^{(s)}_p \rvert$, $1 \leq p \leq k_s$
$n^{(s)}_p$	Dest. partition size, $n^{(s)}_p \equiv \lvert J^{(s)}_p \rvert$, $1 \leq p \leq \ell_s$
$\mathcal{G}^{(s)}_{p,q}$	Subgraphs induced by pth and qth partitions of segment s, i.e., subgraph segment
$\lvert\mathcal{G}^{(s)}_{p,q}\rvert$	Size of subgraphs segment, $\lvert\mathcal{G}^{(s)}_{p,q}\rvert := m^{(s)}_p n^{(s)}_q (t_{s+1} - t_s)$
$\lvert E\rvert^{(s)}_{p,q}$	Number of edges in $\mathcal{G}^{(s)}_{p,q}$
$\rho^{(s)}_{p,q}$	Density of $\mathcal{G}^{(s)}_{p,q}$, $\frac{\lvert E\rvert^{(s)}_{p,q}}{\lvert\mathcal{G}^{(s)}_{p,q}\rvert}$
$H(.)$	Shannon entropy function

Definition 1 (Graph stream) A graph stream $\mathcal{G}$ is a sequence of graphs $G^{(t)}$, i.e.,

$$\mathcal{G} := \{G^{(1)}, G^{(2)}, \ldots, G^{(t)}, \ldots\},$$

which grows indefinitely over time. Each of these graphs links m source nodes to n destination nodes.

For example in Fig. 3.2, the first row plots first three graphs in a graph stream, where $m = 4$ and $n = 3$. Furthermore, the graphs are represented in sparse matrices as shown in the bottom of Fig. 3.2 (a black entry is 1 which indicates an edge between the corresponding nodes; likewise, a white entry is 0).

In general, each graph may be viewed as an $m \times n$ binary adjacency matrix, where rows $1 \leq i \leq m$ correspond to source nodes and columns $1 \leq j \leq n$ correspond to destination nodes. We use sparse representation of the matrix (i.e., only non-zero entries are stored) whose space consumption is similar to adjacency list representation. For the convenience of presentation, we assume the same m and n for all graphs in the graph stream $\mathcal{G}$. However, our algorithms also apply for the graphs with different size by essentially setting some rows and columns to be all zeros in the adjacency matrices.

One of our goals is to track how the structure of the graphs $G^{(t)}$, $t \geq 1$, evolves over time. To that end, we will group consecutive timestamps into segments.

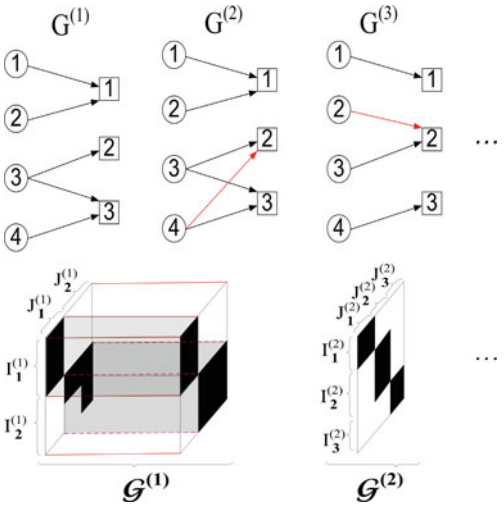


Fig. 3.2 Notation illustration: A graph stream with three graphs in two segments. First graph segment consisting of $G^{(1)}$ and $G^{(2)}$ has two source partitions $I_1^{(1)} = \{1, 2\}$, $I_2^{(1)} = \{3, 4\}$; two destination partitions $J_1^{(1)} = \{1\}$, $J_2^{(1)} = \{2, 3\}$. Second graph segment consisting of $G^{(3)}$ has three source partitions $I_1^{(2)} = \{1\}$, $I_2^{(2)} = \{2, 3\}$, $I_3^{(2)} = \{4\}$; three destination partitions $J_1^{(2)} = \{1\}$, $J_2^{(2)} = \{2\}$, $J_2^{(2)} = \{3\}$

Definition 2 (Graph stream segment) The set of graphs between timestamps t_s and $t_{s+1} - 1$ (inclusive) consist the sth segment $\mathcal{G}^{(s)}$, $s \geq 1$, which has length $t_{s+1} - t_s$,

$$\mathcal{G}^{(s)} := \{G^{(t_s)}, G^{(t_s+1)}, \ldots, G^{(t_{s+1}-1)}\}.$$

Intuitively, a "*graph stream segment*" (or just "*graph segment*") is a set of consecutive graphs in a graph stream. For example in Fig. 3.2, $\mathcal{G}^{(1)}$ is a graph segment consisting of two graph $G^{(1)}$ and $G^{(2)}$.

Next, within each segment, we will partition the source and destination nodes into source partitions and destination partitions, respectively.

Definition 3 (Graph segment partitions) For each segment $s \geq 1$, we partition source nodes into k_s source partitions and destination nodes into ℓ_s destination partitions. The set of source nodes that are assigned into the pth source partition $1 \leq p \leq k_s$ is denoted by $I_p^{(s)}$. Similarly, the set of destination nodes assigned to the qth destination partition is denoted by $J_q^{(s)}$, for $1 \leq q \leq \ell_s$.

The sets $I_p^{(s)}$ partition the source nodes, in the sense that $I_p^{(s)} \cap I_{p'}^{(s)} = \emptyset$ for $p \neq p'$, while $\bigcup_p I_p^{(s)} = \{1, \ldots, m\}$. Similarly, the sets $J_q^{(s)}$ partition the destination nodes. For example in Fig. 3.2, the first graph segment $\mathcal{G}^{(1)}$ has source partitions $I_1^{(1)} = \{1, 2\}$, $I_2^{(1)} = \{3, 4\}$, and destination partitions $J_1^{(1)} = \{1\}$, $J_2^{(1)} = \{2, 3\}$ ($k_1 = 2$,$\ell_1 = 2$). Similarly, we can define source and destination partition for the second graph segment $\mathcal{G}^{(2)}$ ($k_2 = 3$,$\ell_2 = 3$).

3.3.2 Problem Formulation

In this chapter, the ultimate goals are to find communities on the time-evolving graph (along with the *change points*, if any), as well as to compress them incrementally. To achieve that, the following two problems need to be addressed.

Problem 1 (PartitionIdentification) Given a graph stream segment $\mathcal{G}^{(s)}$, how to find the optimal partitions of source and destination nodes such that the encoding cost for $\mathcal{G}^{(s)}$ is minimized.

To achieve this objective, two important sub-questions need to be answered (see Section 3.5.1):

- How to assign the m source and n destination nodes into k_s source and ℓ_s destination partitions?
- How to determine the k_s and ℓ_s?

Problem 2 (TimeSegmentation) Given a graph stream $\mathcal{G}$, how can we incrementally construct graph segments such that the encoding cost for $\mathcal{G}$ is small?

Section 3.5.2 presents the algorithms, where for every new graph $G^{(t)}$ it compares the encoding cost of including $G^{(t)}$ into the current segment vs. that of starting a new segment from timestamp t. We name the whole analytic process, GraphScope.

Next, we will present how to solve both the mining and compression problems on time-evolving graphs using the MDL principle. More specifically, Section 3.4 introduces the encoding objective function; then Section 3.5 presents the algorithm to optimize the proposed objective.

3.4 GraphScope Compression Objective

In this section, we present the encoding scheme of the graph stream and the partitions, which can give us an objective measure of how well a particular compression scheme performs. That is, we assume that we are given some change-points, and the source and destination partitions for each graph segment, and we show how to estimate the compression cost.

3.4.1 Graph Encoding

In this chapter, we represent a graph as a m-by-n binary matrix, where every row or column corresponds to a source or destination node. For example in Fig. 3.2, $G^{(1)}$ is represented as

$$G^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.1}$$

For a given binary matrix, we can store them as a binary string with the length mn along with the two integers m and n. For example, (3.1) can be stored as 1100,0010,0011 (in the column major order) along with two integers 4 and 3.

To further save space, we can adopt some standard lossless compression scheme (such as Huffman coding, arithmetic coding [8]) to encode the binary string, which formally can be viewed as a sequence of realizations of a binomial random variable X. The code length for that is estimated as $mnH(X)$ where $H(X)$ is the entropy of variable X. For notational convenience, we also write that as $mnH(G^{(t)})$. Additionally, three integers need to be stored: the matrix sizes m and n, the number of ones in matrix $|E|$ (the number of edges in the graph). The cost for storing three integers is $\log^\star |E| + \log^\star m + \log^\star n$ bits, where $\log^\star$ is the universal code length

for an integer.[1] Notice that this scheme can be extended to multiple graphs with minor modifications.

More generally, if the random variable X can take values from the set M, with the size $|M|$ (a multinomial distribution), the entropy of X is

$$H(X) = -\sum_{x \in M} p(x) \log p(x),$$

where $p(x)$ is the probability that $X = x$. Moreover, the maximum of $H(X)$ is $\log |M|$ when $p(x) = \frac{1}{|M|}$ for all $x \in M$ (pure random, most difficult to compress); the minimum is 0 when $p(x) = 1$ for a particular $x \in M$ (deterministic, easiest to compress). For the binomial case, if all symbols are all 0 or all 1 in the string, we do not have to store anything because by knowing the number of ones in the string and the sizes of matrix, the receiver is already able to decode the data completely.

With this observation in mind, the goal is to organize the matrix (graph) into some homogeneous sub-matrices with low entropy and compress them separately as we will describe next.

3.4.2 Graph Segment Encoding

Given a graph stream segment $\mathcal{G}^{(s)}$ and its partition assignments, we can precisely compute the cost for transmitting the segment as two parts: (1) *Partition encoding cost*: the model complexity for partition assignments, (2) *Graph encoding cost*: the actual code for the graph segment.

3.4.2.1 Partition Encoding Cost

The description complexity for transmitting the partition assignments for graph segment $\mathcal{G}^{(s)}$ consist of the following terms:

First, we need to send the number of source and destination nodes m and n using $\log^\star m + \log^\star n$ bits. Note that, this term is constant, which has no effect on the choice of final partitions.

Second, we shall send the number of source and destination partitions which is $\log^\star k_s + \log^\star \ell_s$.

Third, we shall send the source and destination partition assignments. To exploit the non-uniformity across partitions, the encoding cost is $m\,H(P) + n\,H(Q)$ where P is a multinomial random variable with the probability $p_i = \frac{m_i^{(s)}}{m}$ $\left(m_i^{(s)}\right.$ is the size of ith source partition, $\left.1 \le i \le k_s\right)$; Q is another multinomial random variable with $q_i = \frac{n_i^{(s)}}{n}$ $\left(\right.$ $n_i^{(s)}$ is the size of ith destination partition, $\left.1 \le i \le \ell_s\right)$.

[1] To encode a positive integer x, we need $\log^\star x \approx \log_2 x + \log_2 \log_2 x + \cdots$, where only the positive terms are retained and this is the optimal length, if the range of x is unknown [25]

For example in Fig. 3.2, the partition sizes for first segment $\mathcal{G}^{(1)}$ are $m_1^{(1)} = m_2^{(1)} = 2$, $n_1^{(1)} = 1$, and $n_2^{(1)} = 2$; the partition assignments for $\mathcal{G}^{(1)}$ costs $-4\left(\frac{2}{4}\log\left(\frac{2}{4}\right) + \frac{2}{4}\log\left(\frac{2}{4}\right)\right) - 3\left(\frac{1}{3}\log\left(\frac{1}{3}\right) + \frac{2}{3}\log\left(\frac{2}{3}\right)\right)$.

In summary, the partition encoding cost for graph segment $\mathcal{G}^{(s)}$ is

$$C_p^{(s)} := \log^\star m + \log^\star n + \log^\star k_s + \log^\star \ell_s + mH(P) + nH(Q), \tag{3.2}$$

where P and Q are multinomial random variables for source and destination partitions, respectively.

3.4.2.2 Graph Encoding Cost

After transmitting the partition encoding, the actual graph segment $\mathcal{G}^{(s)}$ is transmitted as $k_s \ell_s$ subgraph segments. To facilitate the discussion, we define the entropy term for a subgraph segment $\mathcal{G}_{p,q}^{(s)}$ as

$$H\left(\mathcal{G}_{p,q}^{(s)}\right) = -\left(\rho_{p,q}^{(s)} \log \rho_{p,q}^{(s)} + \left(1 - \rho_{p,q}^{(s)}\right) \log\left(1 - \rho_{p,q}^{(s)}\right)\right), \tag{3.3}$$

where $\rho_{p,q}^{(s)} = \frac{|E|_{p,q}^{(s)}}{|\mathcal{G}_{p,q}^{(s)}|}$ is the density of subgraph segment $\mathcal{G}_{p,q}^{(s)}$. Intuitively, it quantifies how difficult it is to compress the subgraph segment $\mathcal{G}_{p,q}^{(s)}$. In particular, if the entire subgraph segment is all 0 or all 1 (the density is exactly 0 or 1), the entropy term becomes 0.

With this, the graph encoding cost is

$$C_g^{(s)} := \sum_{p=1}^{k_s} \sum_{q=1}^{\ell_s} \left(|E|_{p,q}^{(s)} + |\mathcal{G}_{p,q}^{(s)}| \cdot H\left(\mathcal{G}_{p,q}^{(s)}\right)\right), \tag{3.4}$$

where $|E|_{p,q}^{(s)}$ is the number of edges in the (p, q) subgraphs of segment s; $|\mathcal{G}_{p,q}^{(s)}|$ is the size of subgraph segment, i.e., $m_p^{(s)} n_q^{(s)} (t_{s+1} - t_s)$, and $H\left(\mathcal{G}_{p,q}^{(s)}\right)$ is the entropy of the subgraph segment defined in (3.3).

In the subgraph segment $\mathcal{G}_{2,2}^{(1)}$ of Fig. 3.2, the number of edges $|E|_{2,2}^{(1)} = 3+4$, $\mathcal{G}_{2,2}^{(1)}$ has the size $|\mathcal{G}_{2,2}^{(1)}| = 2 \times 2 \times 2$, the density $\rho_{2,2}^{(1)} = \frac{7}{8}$, and the entropy $H\left(\mathcal{G}_{2,2}^{(1)}\right) = -\left(\frac{7}{8}\log\frac{7}{8} + \frac{1}{8}\log\frac{1}{8}\right)$.

Putting everything together, we obtain the segment encoding cost as the following:

Definition 4 (Segment encoding cost)

$$C^{(s)} := \log^\star(t_{s+1} - t_s) + C_p^{(s)} + C_g^{(s)}, \tag{3.5}$$

where $t_{s+1} - t_s$ is the segment length, $C_p^{(s)}$ is the partition encoding cost, $C_g^{(s)}$ is the graph encoding cost.

3.4.3 Graph Stream Encoding

Given a graph stream $\mathcal{G}$, we partition it into a number of graph segments $\mathcal{G}^{(s)}(s \geq 1)$ and compress each segment separately such that the total encoding cost is small.

Definition 5 (Total cost) The total encoding cost is

$$C := \sum_s C^{(s)}, \tag{3.6}$$

where $C^{(s)}$ is the encoding cost for sth graph stream segment.

For example in Fig. 3.2, the encoding cost C up to timestamp 3 is the sum of the costs of two graph stream segments $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$.

Having defined the objective precisely in (3.3) and (3.4), the next step is to search for the optimal partitions and time segmentation. However, finding the optimal solution for this problem is simply NP-hard [9]. Next, we present a heuristic-based search method, GraphScope in Section 3.5 which guarantees to lead to a local optima. From our experiments it often lead to a global optimal solution.

3.5 GraphScope

In this section we describe our method, GraphScope by solving the two problems proposed in Section 3.3.2.

The goal is to find the appropriate number and position of change points, and the number and membership of source and destination partitions so that the cost of (3.6) is minimized. Exhaustive enumeration is prohibitive, and thus we resort to heuristics that we describe next. Note that we drop the subscript s on k_s and ℓ_s whenever it is clear in context.

Specifically, we have three steps: (a) how to find good communities (source and destination partitions), for a given set of (similar) graph snapshots, when we have decided the number of partitions k and l, (b) how to find good values for k and l, and (c) when to declare a time-tick as a *change point* and start a new graph segment. We describe each next.

3.5.1 Partition Identification

Here we explain how to find source and destination partitions for a given graph segment $\mathcal{G}^{(s)}$. In order to do that, we need to answer the following two questions:

– How to find the best partitions given the number of source and destination partitions?
– How to search for the right number of source and destination partitions?

Next, we present the solution for each step.

3.5.1.1 Finding the Best Partitions

Given the number of the best source and destination partitions k and ℓ, we want to regroup sources and destinations into the better partitions. Typically this regrouping procedure is alternating between source and destination nodes. Namely, we update the source partitions with respect to the current destination partitions, and vice versa. More specifically, we alternate the following two steps until it converges:

– *Update source partitions*: for each source (a row of the graph matrix), consider assigning it to the source partition that incurs smallest encoding cost.
– *Update destination partitions*: Once done with a pass over each row, similarly, for each destination (column), consider assigning it to the destination partition that yields the best compression.

The cost of assigning a row to a row group is discussed later (see (3.8)). The pseudocode is listed in Algorithm 1. The complexity of each iteration is either $O(kn)$ for column regrouping or $O(lm)$ for row regrouping. Note that this complexity is independent to the number of graphs in a graph segment. That means as the graph segment grows in time, the computation cost for regrouping remains more or less constant.

Algorithm 1 ReGroup (Graph Segment $\mathcal{G}^{(s)}$; partition size k,ℓ; initial partitions $I^{(s)}$, $J^{(s)}$)

```
Compute density ρ^(s)_{p,q} for all p, q based on I^(s), J^(s). repeat
    forall source s in G^(s) do
        // assign s to the most similar partition
        c is split in ℓ parts
        compute source density p_i for each part
        assign s to source partition with the minimal encoding cost (Equation 3.8).
    Update destination partitions similarly
until no change ;
```

Figure 3.3 illustrates the algorithm in action. The graph consists of two square sub-matrices with the size 150 and 50 plus 1% noise. For $k = \ell = 2$, the algorithm identified the correct partitions in one pass. Notice that the algorithm progressively finds better partitions. The initialization of Algorithm 1 is a crucial step which is discussed separately in Section 3.5.3.

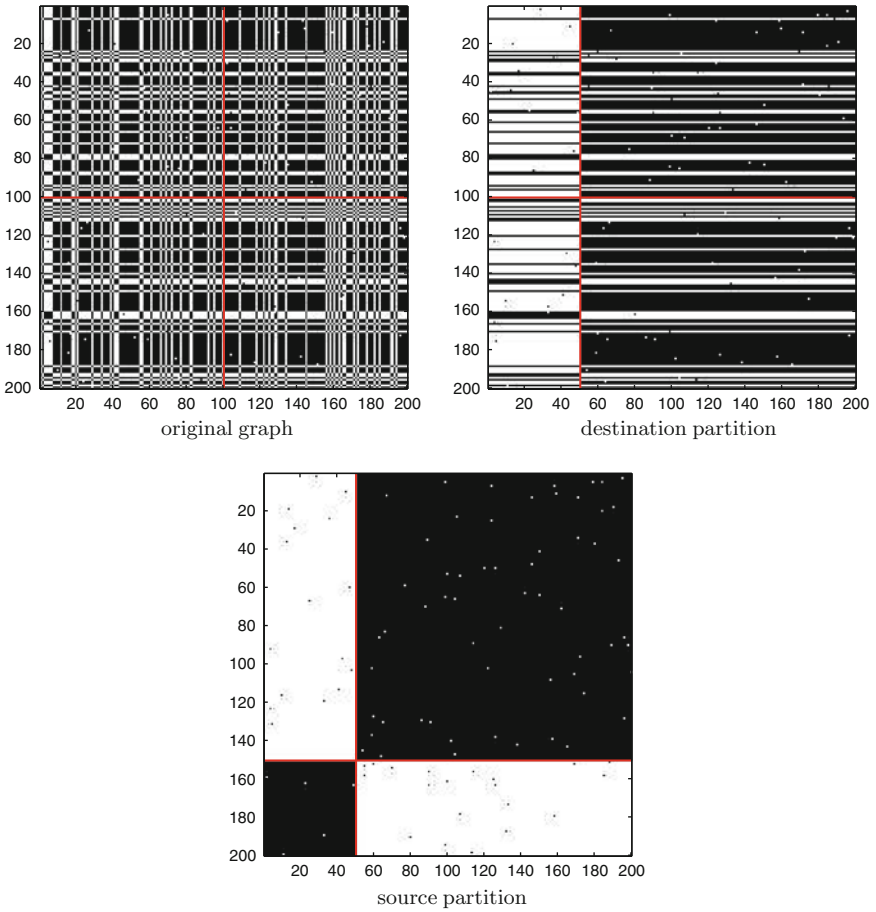


Fig. 3.3 Alternating partition on source and destination nodes on a graph with two communities with size 150 and 50 plus 1% noise. For $k = \ell = 2$, the correct partitions are identified after one pass

3.5.1.2 Determining the Number of Partitions

Given two different k and ℓ, we can easily run Algorithm 1 and choose the ones with a smaller encoding cost. However, the search space for the right k and ℓ is still too large to perform exhaustive tests. We experimented with a number of different heuristics for quickly adjusting k and ℓ and obtained good results with Algorithm 2. The central idea is to do local search around some a priori partition assignments and adjust the number of partitions k and ℓ as well as partition assignments based on the encoding cost. Figure 3.4 illustrates the search process in action. Starting the search with $k = \ell = 1$, it successfully finds the correct number of partitions for this graph with three sub-matrices with size 100, 80, and 20.

Algorithm 2 SEARCHKL (Graph Segment $\mathcal{G}^{(s)}$; initial partition size k,ℓ; initial partitions $I^{(s)}$, $J^{(s)}$)

```
repeat
    // try to merge source partitions
    repeat
        Find the source partition pair (x, y) s.t. merging x and y gives smallest encoding
        cost for G^(s).
        if total encoding decrease then merge x,y
    until no more merge ;
    // try to split source partition
    repeat
        Find source partition x with largest average entropy per node.
        foreach source s in x do
            if average entropy reduces without s then
                assign s to the new partition
        ReGroup(G^(s), updated partitions)
    until no decrease in encoding cost ;
    Search destination partitions similarly
until no changes ;
```

3.5.1.3 Cost Computation for Partition Assignments

Here we present the details of how to compute the encoding cost of assigning a node to a particular partition. Our discussion focuses on assigning of a source node to a source partition, while the assignment for a destination node is simply symmetric.

Recall a graph segment $\mathcal{G}^{(s)}$ consists of $(t_{s+1} - t_s)$ graphs, $G^{(t_s)}, \ldots, G^{(t_{s+1}-1)}$. For example in Fig. 3.2, $\mathcal{G}^{(1)}$ consists of two graphs, $G^{(1)}$ and $G^{(2)}$. Likewise, every source node in a graph segment $\mathcal{G}^{(s)}$ consists $(t_{s+1} - t_s)$ sets of edges to these $(t_{s+1} - t_s)$ graphs. Therefore, the total number of possible edges out of one source node in $\mathcal{G}^{(s)}$ is $(t_{s+1} - t_s)n$.

Furthermore, the destination partitions $J_i^{(s)}$ split the destination nodes into ℓ disjoint sets with size $n_i^{(s)}$ $(1 \le i \le \ell, \sum_i n_i^{(s)} = n)$. For example, $\mathcal{G}^{(1)}$ of Fig. 3.2 has two destination partitions ($\ell = 2$), where the first destination partition $J_1^{(1)} = \{1\}$ and the second destination partition $J_2^{(1)} = \{2, 3\}$.

Similarly, all the edges from a single source node in graph segment $\mathcal{G}^{(s)}$ are also split into these ℓ sets. In $\mathcal{G}^{(1)}$ of Fig. 3.2, the edges from the 4th source node are split into two sets, where the first set $J_1^{(1)}$ consists of 0 edges and the second set $J_2^{(1)}$ consists of 3 edges.[2]

More formally, the edge pattern out of a source node is generated from ℓ binomial distributions $\mathbf{p}_i (1 \le i \le \ell)$ with respect to ℓ destination partitions. Note that $\mathbf{p}_i(1)$ is just the density of the edges from that source node to the destination partition $J_i^{(s)}$,

[2] One edge from 4 to 3 in $G^{(1)}$, two edges from 4 to 2 and 3 in $G^{(2)}$ in Fig. 3.2.

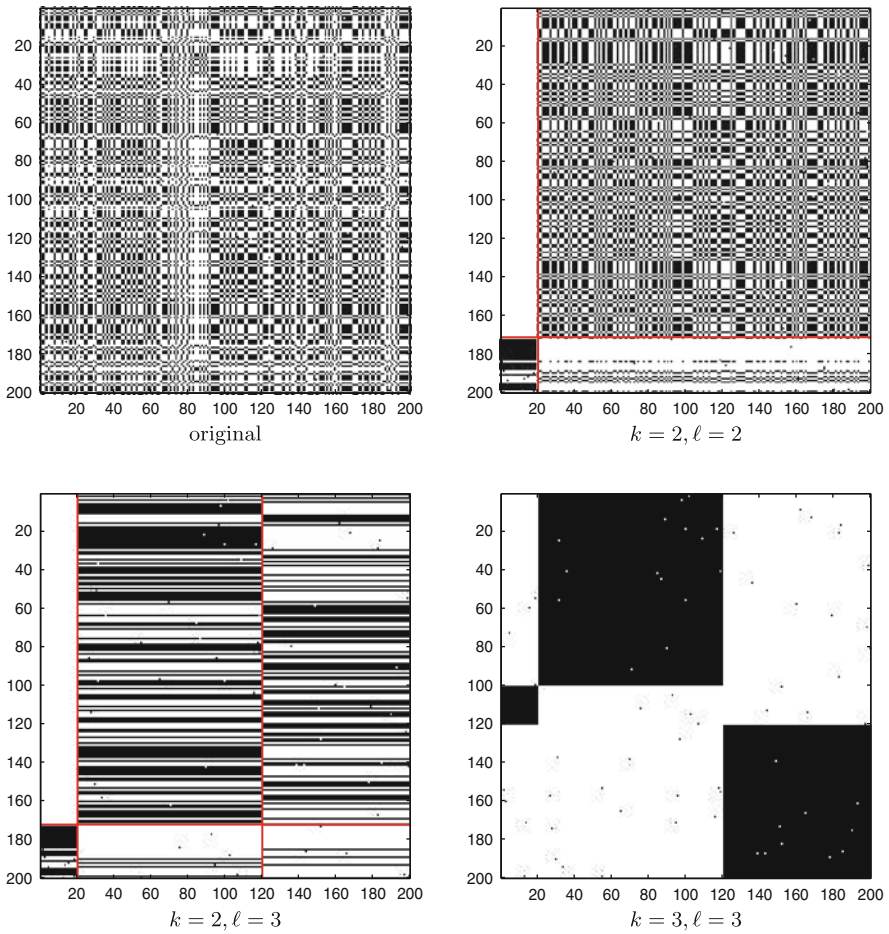


Fig. 3.4 Search for best k and ℓ for a graph with three communities with size 100, 80, 20 plus 1% noise. The algorithm progressively improves the partition quality (reduces the compression cost) by changing the k and ℓ

and $\mathbf{p}_i(0) = 1 - \mathbf{p}_i(1)$. In $\mathcal{G}^{(1)}$ of Fig. 3.2, the 4th source node has $\mathbf{p}_1(1) = 0$ since there are 0 edges from 4 to $J_1^{(1)} = \{1\}$, and $\mathbf{p}_1(1) = \frac{3}{4}$ since 3 out of 4 possible edges from 4 to $J_1^{(2)} = \{2, 3\}$.

Using this "true" distribution, the encoding cost of the source node in the graph segment $\mathcal{G}^{(s)}$ is

$$C(\mathbf{p}) = (t_{s+1} - t_s)n \sum_{i=1}^{\ell} H(\mathbf{p}_i), \tag{3.7}$$

where $(t_{s+1} - t_s)$ is the number of graphs in the graph segment, n is the number of possible edges out of a source node for each graph,[3] $H(\mathbf{p}_i) = \sum_{x=\{0,1\}} \mathbf{p}_i(x) \log \mathbf{p}_i(x)$ is the entropy for the given source node.

In $\mathcal{G}^{(1)}$ of Fig. 3.2, the number of graphs is $t_{s+1} - t_s = 3 - 1 = 2$; the number of possible edges out of the 4th source node $n = 3$; therefore, the 4th source node costs $2 \times 3 \times \left(0 + \frac{3}{4} \log \frac{3}{4} + \frac{1}{4} \log \frac{1}{4}\right) = 2.25$. Unfortunately, this is not practical to do so for every source node, because the model complexity is too high. More specifically, we have to store additional $m\ell$ integers in order to decode all source nodes.

The practical option is to group them into a handful number of source partitions and to encode/decode one partition at a time instead of one node at a time. Similar to a source node, the edge pattern out of a source partition is also generated from ℓ binomial distributions $\mathbf{q}_i$ $(1 \leq i \leq \ell)$. Now we encode the ith source node based on the distribution $\mathbf{q}_i$ for a partition instead of the "true" distribution $\mathbf{p}_i$ for the node. The encoding cost is

$$C(\mathbf{p}, \mathbf{q}) = (t_{s+1} - t_s) m \sum_{i=1}^{\ell} H(\mathbf{p}_i, \mathbf{q}_i), \tag{3.8}$$

where $H(\mathbf{p}_i, \mathbf{q}_i) = \sum_{x=\{0,1\}} \mathbf{p}_i(x) \log \mathbf{q}_i(x)$ is the cross-entropy. Intuitively, the cross-entropy is the average encoding cost when using the distribution $\mathbf{q}_i$ instead of the "true" distribution $\mathbf{p}_i$. In $\mathcal{G}^{(1)}$ of Fig. 3.2, the cost of assigning the 4th node to second source partition $I_2^{(1)}$ is $2 \times 3 \times \left(0 + \frac{3}{4} \log \frac{7}{8} + \frac{1}{4} \log \frac{1}{8}\right) = 2.48$ which is slightly higher than using the true distribution that we just computed (2.25). However, the model complexity is much lower, i.e., $k\ell$ integers are needed instead of $m\ell$.

3.5.2 Time Segmentation

So far, we have discussed how to partition the source and destination nodes given a graph segment $\mathcal{G}^{(s)}$. Now we present the algorithm how to construct the graph segments incrementally when new graph snapshots arrive every time-tick. Intuitively, we want to group "similar" graphs from consecutive timestamps into one graph segment and encode them altogether. For example, in Fig. 3.2, graphs $G^{(1)}$, $G^{(2)}$ are similar (only one edge difference), and therefore we group them into one graph segment, $\mathcal{G}^{(1)}$. On the other hand, $G^{(3)}$ is quite different from the previous graphs, and hence we start a new segment $\mathcal{G}^{(2)}$ whose first member is $G^{(3)}$.

The driving force here is still the compression cost. More specifically, the algorithm will combine the incoming graph with the current graph segment if there is a compression benefit, otherwise we start a new segment with that graph. The meta-algorithm is listed in Algorithm 3. Figure 3.5 illustration the algorithm in action. A graph stream consists of three graphs, where $G^{(1)}$ and $G^{(2)}$ have two groups of size

[3] $(t_{s+1} - t_s)n$ is the total number of possible edges of a source node in the graph segment

Algorithm 3 GRAPHSCOPE (Graph Segment $\mathcal{G}^{(s)}$; Encoding cost c_o; New Graph $G^{(t)}$

output: updated graph segment, new partition assignment $I^{(s)}$, $J^{(s)}$
Compute new encoding c_n of $\mathcal{G}^{(s)} \bigcup \{G^{(t)}\}$
Compute encoding cost c for just $G^{(t)}$
`// check if there is any compression benefit`
if $c_n - c_o < c$ **then**
  `// add` $G^{(t)}$ `in` $\mathcal{G}^{(s)}$
  $\mathcal{G}^{(s)} \leftarrow \mathcal{G}^{(s)} \bigcup \{G^{(t)}\}$
  searchKL for updated $\mathcal{G}^{(s)}$
else
  `// start a new segment from` $G^{(t)}$
  $\mathcal{G}^{(s+1)} := \{G^{(t)}\}$
  searchKL for new $\mathcal{G}^{(s+1)}$

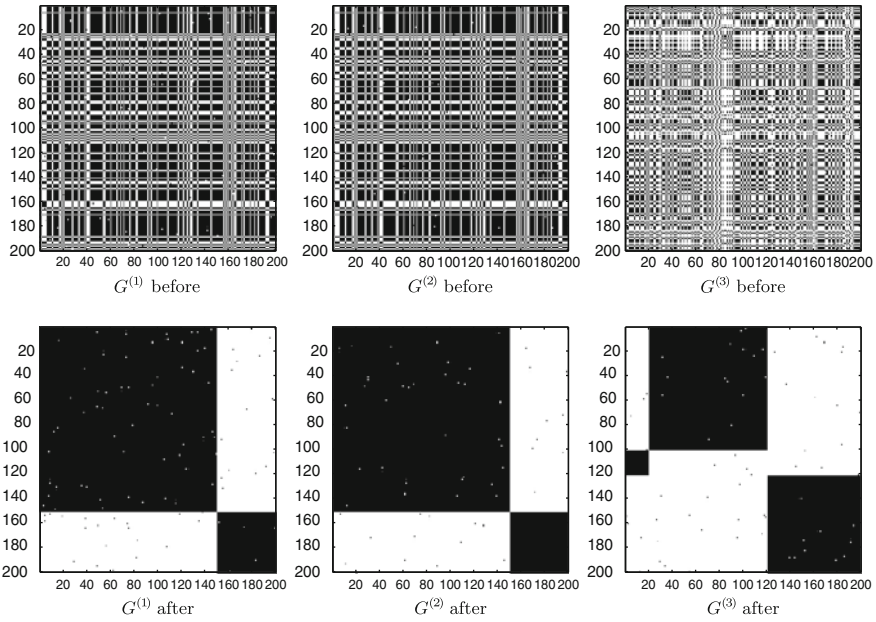


Fig. 3.5 A graph stream with three graphs: The same communities appear in graph $G^{(1)}$ and $G^{(2)}$; therefore, they are grouped into the same graph segment. However, $G^{(3)}$ has different community structure, therefore, a new segment starts from $G^{(3)}$

150 and 50, $G^{(3)}$ three groups of size 100, 80 and 20, and every graph contains 1% noise. The algorithm decides to group $G^{(1)}$, $G^{(2)}$ into the first graph segment, and put $G^{(3)}$ into another. During this process, the correct partitions are also identified as show in the bottom of Fig. 3.5. Furthermore, within each segment, the correct partition assignments are identified.

3.5.3 Initialization

Once we decide to start a new segment, how should we initialize the parameters of our algorithms? There are several ways to do the initialization. Trading-off convergence speed versus compression quality, we propose and study two such heuristics:

Fresh-Start: One option is to start from a small k and ℓ, typically, $k = 1$ and $\ell = 1$, and progressively increase them as well as regroup sources and destinations into proper partitions. From our experiments, this scheme is very effective in leading to a good result. In terms of computational cost, it is relatively fast since we start with small k and ℓ.

Another option is to start with large k and ℓ and to try to merge them. However, there are two big disadvantages for doing that: (1) *computationally expensive* since the number of partitions are large to start with; (2) *local minimum*, starting with large number of partitions often lead to complex search space and local minimum. For these reasons, we recommend to search from small k and ℓ.

Resume: For time-evolving graphs, consecutive graphs often have a strong similarity. We can leverage this similarity into the search process by starting from old partitions. More specifically, we initialize the k_{s+1} and ℓ_{s+1} with the k_s and ℓ_s. Additionally, we assign $I^{(s+1)}$ and $J^{(s+1)}$ as $I^{(s)}$ and $J^{(s)}$. The *Resume* scheme often lead to much faster convergence when the consecutive graphs are similar as shown in Section 3.6.

3.6 Experiment Evaluation

In this section, we will evaluate both mining and compression aspects of GraphScope using several real, large graph datasets. We first describe the data set specification in Section 3.6.1. Then we present our experiments, which are designed to answer the following questions, for both our variations *fresh-start* and *resume* :

- *Mining Quality*: How good is our method in terms of finding meaningful change points and communities (Section 3.6.2).
- *Compression*: What is the compression ratio it can achieve (Section 3.6.3).
- *Speed*: How fast is it, and how does it scaleup (Section 3.6.4).

Finally, we present some additional mining observations that our method leads to. Notice that we do not compare with other methods for two reasons: First, to the best of our knowledge, there are no clustering methods with the MDL principle for time-evolving graphs, which make it unfair to compare with other methods on compression cost. Second, most published methods are not parameter-free, and it is unclear how we should choose their parameters (number of partitions,

threshold for graph similarity and so on). This is actually one of the strong points of GraphScope, because it is fully automatic, and, as we show, still able to find meaningful communities and change points.

3.6.1 Data Sets

In this section, we describe all the data sets in our experiments (Table 3.2).

Table 3.2 Data set summary

name	m-by-n	avg.$\lvert E\rvert$	time T
NETWORK	29K-by-29K	12K	1, 222
ENRON	34k-by-34k	15K	165
CELLPHONE	97-by-3764	430	46
DEVICE	97-by-97	689	46
TRANSACTION	28-by-28	132	51

3.6.1.1 The NETWORK Flow Data Set

The traffic trace consists of TCP flow records collected at the backbone router of a class-B university network. Each record in the trace corresponds to a directional TCP flow between two hosts with timestamps indicating when the flow started and finished. With this traffic trace, we use a window size of 1 h to construct the source–destination graph stream. Each graph is represented by a sparse adjacency matrix with the rows and the columns corresponding to source and destination IP addresses, respectively. The edge in a graph $G^{(t)}$ means that there exists TCP flows (packets) sent from the ith source to the jth destination during the tth hour. The graphs involve $m = n = 21{,}837$ unique campus hosts (the number of source and destination nodes) with an average over 12 K distinct connections (the number of edges). The total number of timestamps T is 1,222. Figure 3.6a shows an example of superimposing[4] all source–destination graphs in one time segment of 18 h. Every row/column corresponds to a source/destination; the dot there indicates there is at least a packet from the source to the destination during that time segment. The graphs are correlated, with most of the traffic to or from a small set of server-like hosts.

GraphScope automatically exploits the sparsity and correlation by organizing the sources and destinations into homogeneous groups as shown in Fig. 3.6b.

3.6.1.2 The ENRON Email Data Set

This consists of the email communications in Enron Inc. from January 1999 to July 2002 (http://www.cs.cmu.edu/enron/). We construct sender-to-recipient graphs on a

[4] Two graphs are superimposed together by taking the union of their edges.

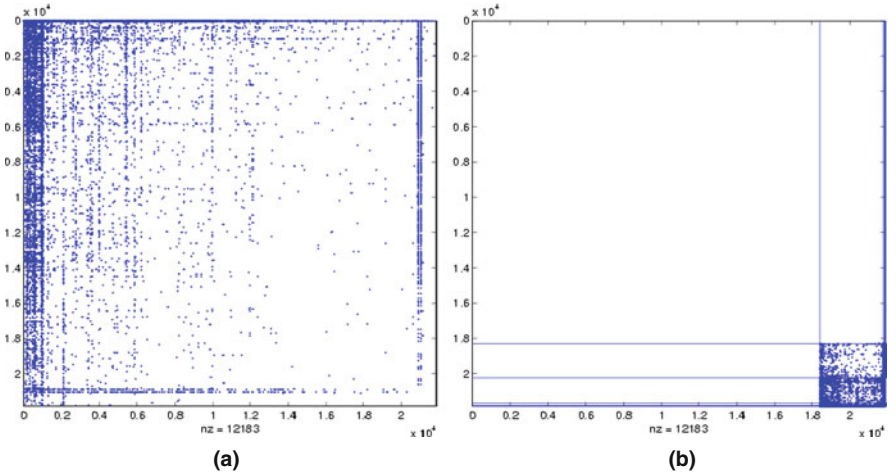


Fig. 3.6 NETWORK before and after GraphScope for the graph segment between January 7 1:00, 2005 and January 7 19:00, 2005. GraphScope successfully rearrange the sources and destinations such that the sub-matrices are much more homogeneous. (**a**) before; (**b**) after

weekly basis. The graphs have $m = n = 34{,}275$ senders/recipients (the number of nodes) with an average 1,479 distinct sender–receiver pairs (the number of edges) every week.

Like the NETWORK data set, the graphs in ENRON are also correlated, where GraphScope can significantly compress the data by reorganizing the graph into homogeneous partitions (see the visual comparison in Fig. 3.7).

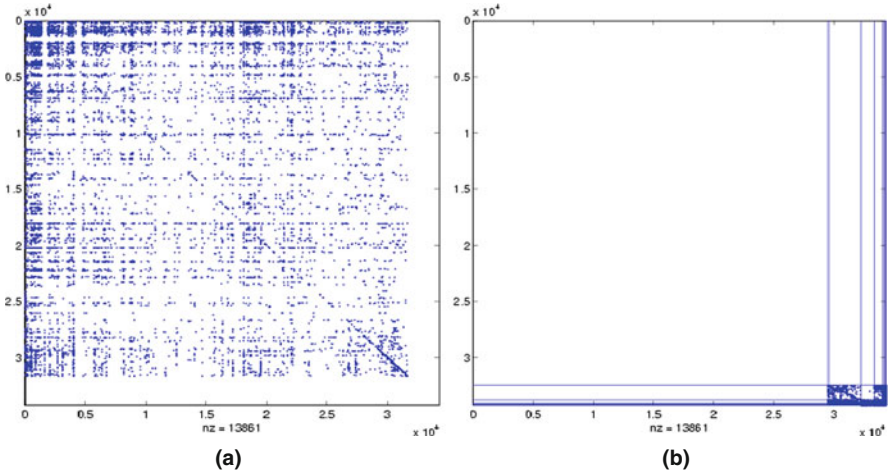


Fig. 3.7 ENRON before and after GraphScope for the graph segment of week 35, 2001, to week 38, 2001. GraphScope can achieve significant compression by partitioning senders and recipients into homogeneous groups. **(a)** before; **(b)** after

3.6.1.3 The Cellphone Communication Data Set

The CELLPHONE data set records the cellphone activity for $m = n = 97$ users from two different labs in MIT (http://reality.media.mit.edu/download.php). Each graph snapshot corresponds to a week, from January 2004 to May 2005. We thus have $T = 46$ graphs, one for each week, excluding weeks with no activity.

We plot the superimposed graphs of 38–42 weeks in 2004 at Fig. 3.8a, which looks much more random than NETWORK and ENRON. However, GraphScope is still able to extract the hidden structure from the graph as shown in Fig. 3.8b, which looks much more homogeneous (more details in Section 3.6.2).

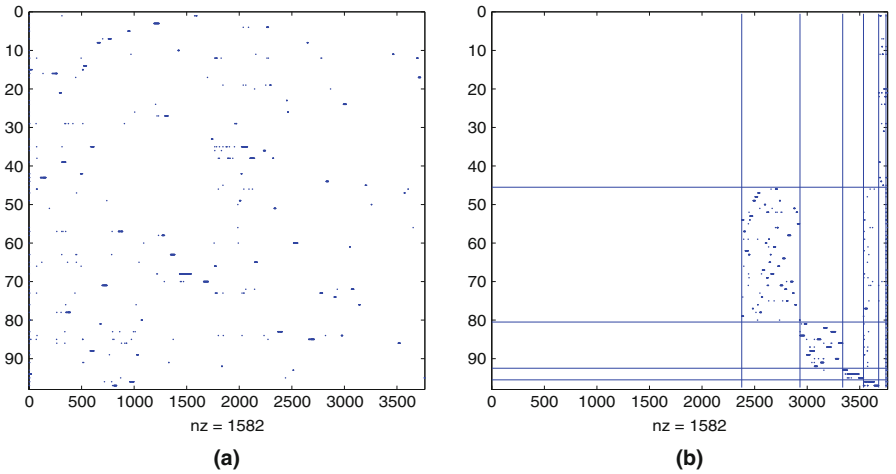


Fig. 3.8 CELLPHONE before and after GraphScope, for the period of week 38–42 in 2004. **(a)** before; **(b)** after

3.6.1.4 The Bluetooth Device Communication Data Set

DEVICE data set is constructed on the same 97 users whose cellphones perform periodic Bluetooth scan for nearby phones and computers. The goal is to understand people's behavior from their proximity information to others. Figure 3.9a plots the superimposed user-to-user graphs for one time segment where every dot indicates that the two corresponding users are physically near to each other. Note that first row represents all the other devices that do not belong to the 97 users (mainly laptop computers, PDAs, and other people's cellphone). Figure 3.9b shows the resulting users clusters of that time segment, where cluster structure is revealed (see Section 3.6.2 for details).

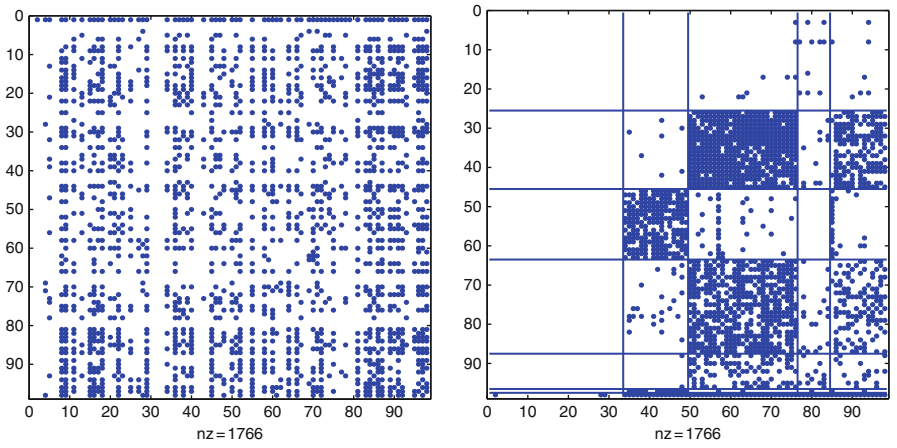


Fig. 3.9 DEVICE before and after GraphScope for the time segment between week 38, 2004 and week 42, 2004. Interesting communities are identified. **(a)** before; **(b)** after

3.6.1.5 The Financial Transaction Data Set

The TRANSACTION data set has $m = n = 28$ accounts of a company, over 2,200 days. An edge indicates that the source account had funds transferred to the destination account. Each graph snapshot covers transaction activities over a window of 40 days, resulting in $T = 51$ time-ticks for our data set.

Figure 3.10a shows the transaction graph for one timestamp. Every black square at the (i, j) entry in Fig. 3.10a indicates there is at least one transaction debiting the ith account and crediting the jth account. After applying GraphScope on that timestamp (see Fig. 3.10b), the accounts are organized into very homogeneous groups with some exceptions (more details in Section 3.6.2).

3.6.2 Mining Case Studies

Now we qualitatively present the mining observation on all the data sets. More specifically, we illustrate that (1) source and destination groups correspond to semantically meaningful clusters; (2) the groups evolve over time; (3) time segments indicate interesting change points

3.6.2.1 NETWORK: Interpretable Groups

Despite the bursty nature of network traffic, GraphScope can successfully cluster the source and destination hosts into meaningful groups. Figure 3.11a,b show the active source and destination nodes organized by groups for two different time

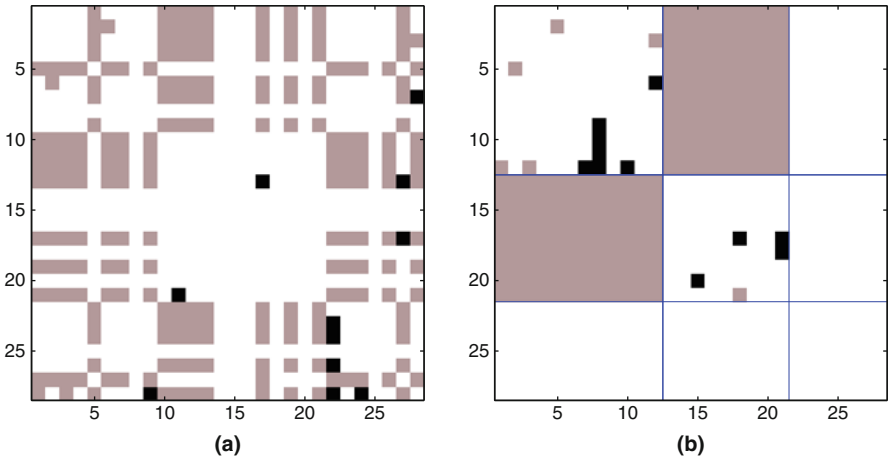


Fig. 3.10 `TRANSACTION` before and after GraphScope for a time segment of 5 months. GraphScope is able to group accounts into partitions based on their types. **(a)** before; **(b)** after

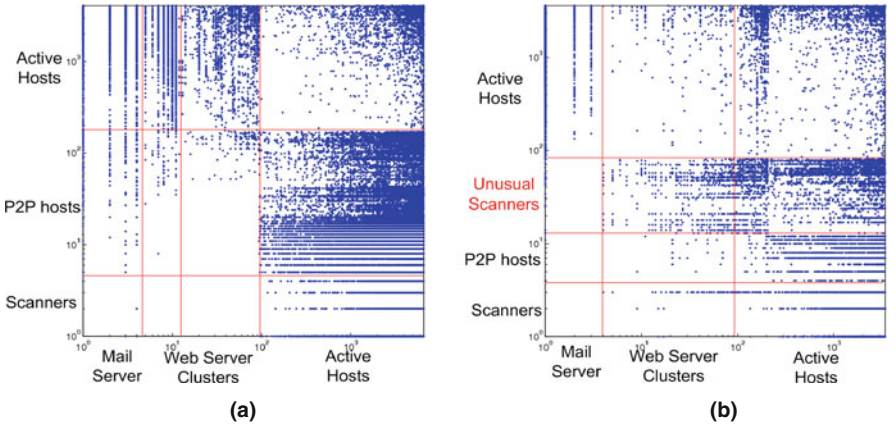


Fig. 3.11 `NETWORK` zoom-in (log–log plot): (**a**) Source nodes are grouped into active hosts and security scanning program; destination nodes are grouped into active hosts, clusters, web servers, and mail servers. (**b**) On a different time segment, a group of unusual scanners appears, in addition to the earlier groups

segments. Note that Fig. 3.11 is plotted in log–log scale in order to visualize those small partitions. For example, source nodes are grouped into (1) active hosts which talk to a small number of hosts, (2) P2P hosts that scan a number of hosts, and (3) security scanning hosts[5] which scans many hosts. Similarly, destination hosts are grouped into (1) active hosts, (2) cluster servers at which many students login

[5] The campus network is constantly running some port-scanning program to identify potential vulnerability of the in-network hosts.

remotely to work on different tasks, (3) Web servers which hosts the Web sites of different schools, and (4) mail servers that have the most incoming connections. The main difference between Fig. 3.11a and b is that a source group of unusual scanners emerges in (b), where GraphScope can automatically identify the change and decide to split into two time segments.

3.6.2.2 CELLPHONE: Evolving Groups

As in NETWORK, we also observe meaningful groups in CELLPHONE. Figure 3.12 a illustrate the calling patterns in fall semester 2004, where two strong user partitions (G1 and G2) exist, the dense small partition G3 is the service call in campus. Figure 3.12b illustrate the calling patterns changed from fall semester to winter break.

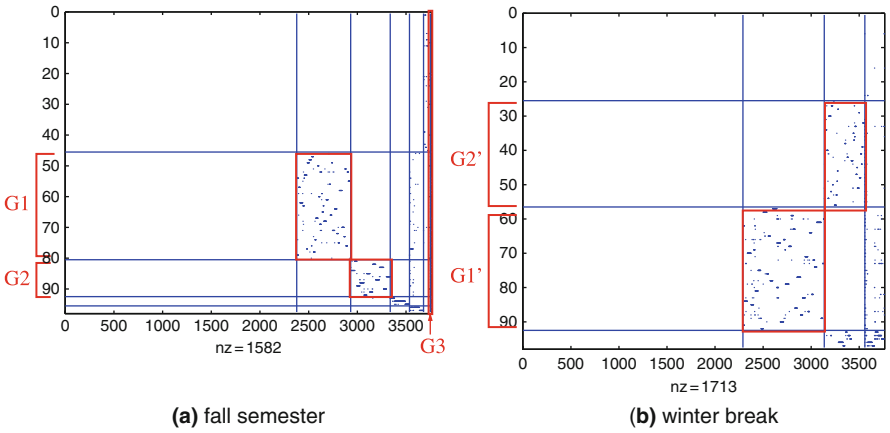


Fig. 3.12 CELLPHONE: (**a**) Two calling groups appear during the fall semester; (**b**) Call groups changed in the winter break. The change point corresponds to the winter break

3.6.2.3 DEVICE: Evolving Groups

Similarly, the group evolving behavior is also observed in the DEVICE data set. In particular, two dense partitions appear in Fig. 3.13a: after inspecting the user ids and their attributes, we found that the users in group $U1$ are all from the same school with similar schedule, probably taking the same class; the users in $U2$ all work in the same lab. In a later time segment (see Fig. 3.13b), the partition $U1$ disappeared, while the partition $U2$ is unchanged.

3.6.2.4 TRANSACTION

As shown in Fig. 3.10b, GraphScope successfully organizes the 28 accounts into 3 different partitions; after closer inspection, these groups correspond to the

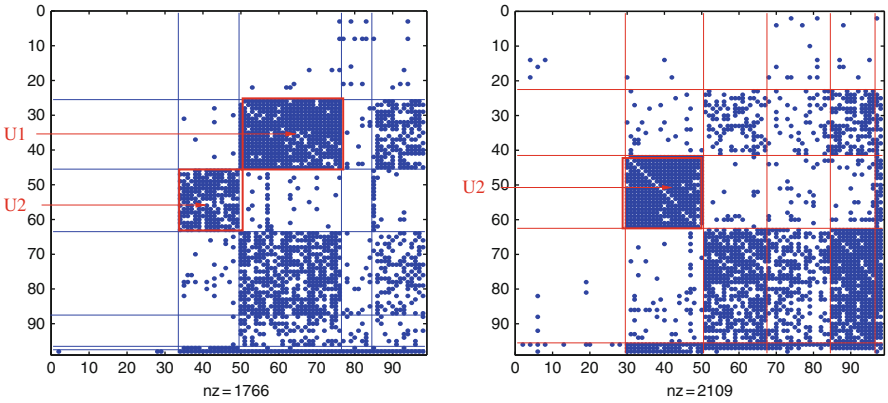


Fig. 3.13 DEVICE: (**a**) Two groups are prominent. Users in $U1$ are all from the same school with similar schedule possibly taking the same class; Users in $U2$ are all working in the same lab. (**b**) $U1$ disappears in the next time segment, while $U2$ remains unchanged

different functional groups of the accounts (like "marketing", "sales").[6] In Fig. 3.10b, the interaction between first source partition (first row) and second destination partition (second column) correspond to mainly the transactions from assets accounts to liability and revenue accounts, which obeys the common business practice.

3.6.2.5 ENRON: Change Point Detection

The source and destination partitions usually correspond to meaningful clusters for the given time segment. Moreover, the time segments themselves usually encode important information about changes. Figure 3.1 plots the encoding cost difference between incorporating the new graph into the current time segment vs. starting a new segment. The vertical lines on Fig. 3.1 are the top 10 splits with largest cost savings when starting a new segment, which actually correspond to the key events related to Enron Inc. Moreover, the intensity in terms of magnitude and frequency dramatically increases around January 2002 which coincides with several key incidents such as the investigation on document shredding and the CEO resignation.

[6] Due anonymity requirements, the account types are obfuscated.

3.6.3 Compression Evaluation

Methods for Comparison: For comparison of the space savings, we consider the following methods:

- *Original*: the space for the original graphs, uncompressed graphs, stored as edges
- *Compression*: global compression estimate for the original graphs, i.e., $mnH(G^{(t)})$
- *Resume*: the encoding cost using GraphScope with the resume heuristics
- *Fresh-Start*: the encoding cost using *resume* GraphScope with the fresh restart heuristics

Performance Metrics: The performance metrics is *Relative Encoding cost:* the ratio of the encoding cost of a graph vs. the space for storing that graph in sparse matrix.

Compression Benefit: We compare two versions of GraphScope, "*fresh-start*" and "*resume*," against the global compression estimate and the space requirement for the original graphs stored as sparse matrices. Figure 3.14 shows that both *fresh-start* and *resume* GraphScope achieve great compression gain (less than 4% of the original space), which is even better than the global compression on the graphs (the 3rd bar for each data set). Our two variations require about the same space.

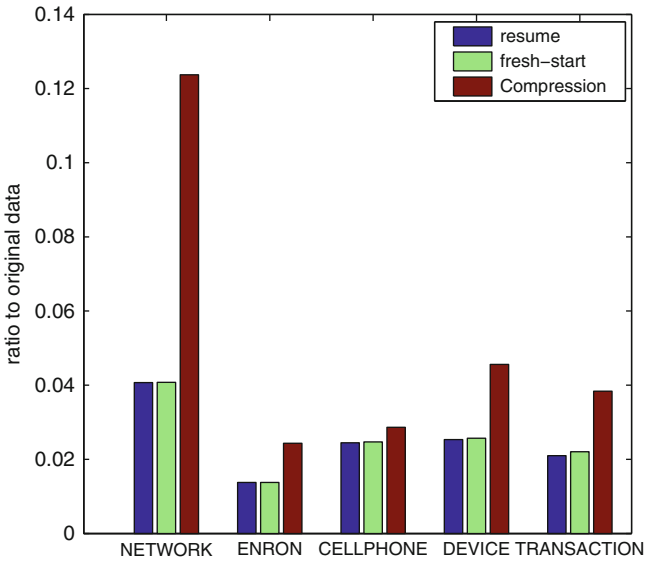
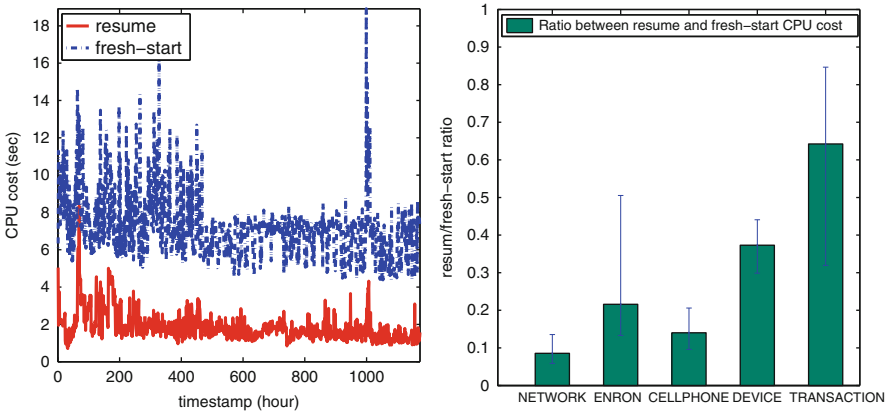


Fig. 3.14 Relative encoding cost: Both *resume* and *fresh-start* methods give over an order of magnitude space saving compared to the raw data and are much better than global compression on the raw data

3.6.4 Speed and Scalability

For the CPU time comparison, we include *fresh-start* and *resume*. The performance metrics is *relative CPU cost* which is the ratio between CPU cost of the *resume* method vs. that of the *fresh-start* method.

As shown in Fig. 3.15a for NETWORK (similar result are achieved for the other data sets, hence omitted), the CPU cost per timestamp/graph is stable over time for both *fresh-start* and *resume* GraphScope, which suggests that both proposed methods are scalable for streaming environments.



(a) CPU cost (NETWORK) **(b)** relative CPU

Fig. 3.15 CPU cost: (**a**) The CPU costs for both *resume* and *fresh start* GraphScope are stable over time; (**b**) *resume* GraphScope is much faster than *fresh start* GraphScope on the same data sets (the error bars give 25 and 75% quantiles)

Furthermore, *resume* GraphScope is much faster than the *fresh-start* one as plotted in Fig. 3.15b, especially for large graphs such as in NETWORK. There, *resume* GraphScope only uses 10% of CPU time compared to *fresh-start* one.

3.6.5 Additional Observations

3.6.5.1 Partition Changing Rate

We define a pair of nodes (i, j) as *inconsistent* if i and j belong to the same partition in $G^{(t)}$ but not in $G^{(t+1)}$. The changing rate between $G^{(t)}$ and $G^{(t+1)}$ is the percentage of inconsistent pairs between two graphs.

Figure 3.16 shows the average changing rate of source and destination partitions for both *fresh-start* and *resume* GraphScope, respectively. Note that the average changing rate is small between two consecutive timestamps, which confirms the heuristics used for *resume* GraphScope, i.e., using the previous partition assignment as initialization for the current graph.

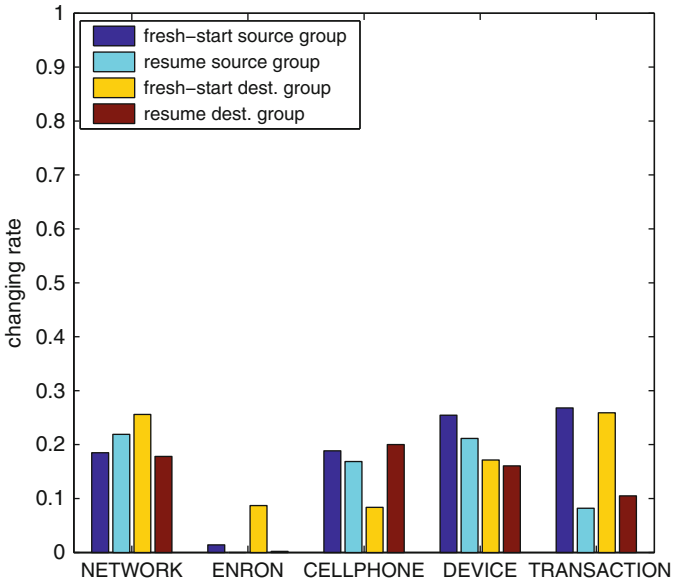


Fig. 3.16 Group changing rate: Overall, the partitions change relatively small for all 5 data sets which explains the benefit for using *resume* GraphScope

3.6.5.2 Time Evolving Aspect

Encoding cost of time segments can illustrate different dynamics in the real graph streams. For example, Fig. 3.17a,d and e show more or less constant trends with stochastic variation, while Fig. 3.17b, and c show higher intensity in the middle of time intervals. In particular, The high intensity in Fig. 3.17b is due to the highly volatile communication in `ENRON` during that period (the end of 2001 and early 2002).

3.7 Discussion and Conclusion

We propose GraphScope, a system to mine and compress streams of graphs. Our method has all the desired properties:

- It is rigorous and automatic, with no need for user-defined parameters, like numbers of communities, thresholds of similarities. Instead, it uses lossless compression and the Minimum Description Language (MDL) principle to decide how to form communities and when to modify them.
- It achieves excellent compression ratios.
- It is fast and scalable, carefully designed to work for a streaming setting.

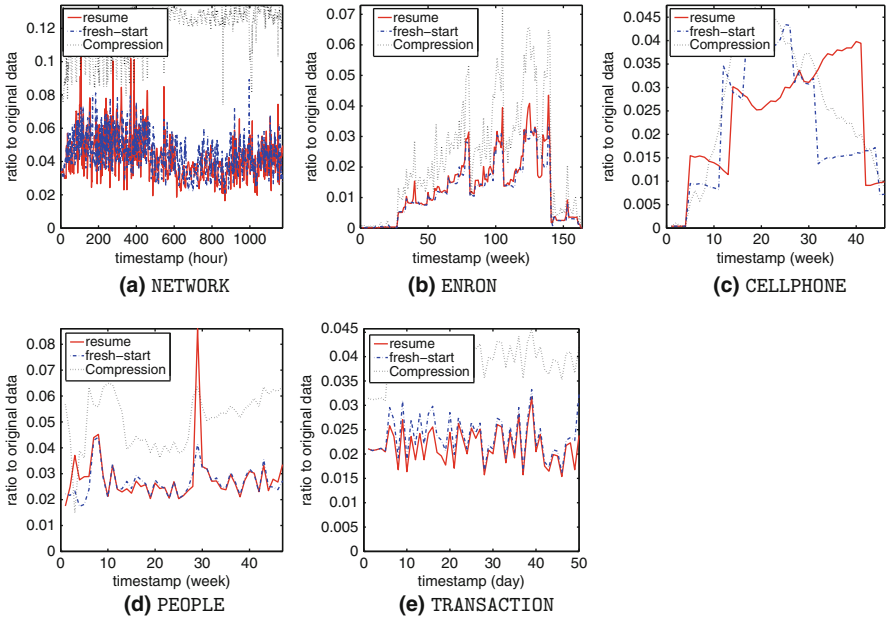


Fig. 3.17 Relative encoding cost over time. **(a)** NETWORK; **(b)** ENRON **(c)** CELLPHONE **(d)** PEOPLE **(e)** TRANSACTION

– It is effective, discovering meaningful communities and meaningful transition points, as shown on our multiple, real data sets, like the major timestamps in the ENRON data set.

The complexity of GraphScope is linear to the number of nodes in the graphs, and independent to the number of timestamps in a graph segment, which means the running will not increase as the graph segment grows. GraphScope treats source and destination node independently in order to provide a framework to deal with more general graphs. However, if the data have the same source and destination nodes, a simple constraint can be enforced to put the corresponding source/destination node to the same cluster.

We also present experiments on several real data sets, spanning 500 GB (200 MB after processing). The data sets were from diverse applications (university network traffic, email from the Enron company, Cellphone call logs, and Bluetooth connections from MIT). Because of its generality and its careful theoretical underpinnings, GraphScope is able to find meaningful groups and patterns in all the above settings, without any specific fine-tuning on our side.

Future research directions include extensions to create hierarchical groupings, both of the communities and of the time segments. GraphScope currently does the time segmentation in a simple online fashion, which does not guarantee the optimality. However, if the streaming constraint is relaxed, a dynamic programming algorithm can be developed to generate the optimal segmentation.

References

1. C. C. Aggarwal and P. S. Yu. Online analysis of community evolution in data streams. In *SDM*, 2005.
2. R. Agrawal, R. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance with a hippocratic database. In *VLDB*, Toronto, Ontario, Canada, 2004.
3. B. Babcock, M. Datar, and R. Motwani. Sampling from a moving window over streaming data. In *SODA*, 2002.
4. L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *KDD*, pages 44–54, 2006.
5. D. Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *PKDD*, pages 112–124, 2004.
6. D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *KDD*, pages 79–88. ACM Press, 2004.
7. G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *TKDE*, 15(3), 2003.
8. T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, 1991.
9. I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *KDD*, pages 89–98, 2003.
10. P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. In *SIAM Journal on Computing*, 36(1):184–206, 2006.
11. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999.
12. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explorations Newsletter*, 3(2), 2002.
13. M. Garofalakis and P. B. Gibbons. Probabilistic wavelet synopses. *ACM Transactions on Database System*, 29(1), 2004.
14. G. H. Golub and C. F. V. Loan. *Matrix Computation*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.
15. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, 2000.
16. G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.
17. E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *KDD*, pages 206–215, New York, NY ACM Press, 2004.
18. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Extracting large-scale knowledge bases from the web. In *VLDB*, Edinburgh, Scotland, 1999.
19. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *SIGKDD*, 2005.
20. Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. Tseng. Analyzing communities and their evolutions in dynamics networks. *ACM Transactions on Knowledge Discovery from Data (TKDD), special issue on Social Computing, Behavioral Modeling*, 3, 2009.
21. Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher. Metafac: Community discovery via relational hypergraph factorization. In *KDD*, 2009.
22. S. Muthukrishnan. Data streams: algorithms and applications, *Foundations and Trends. in Theoretical Computer Science*, 1, 2005.
23. A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2001.
24. S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, Trondheim, Norway, 2005.
25. J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11(2):416–431, 1983.

26. J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: Dynamic tensor analysis. In *KDD*, pages 374–383, 2006.
27. J. Sun, Y. Xie, H. Zhang, and C. Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*, 2007.
28. Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, Hong Kong, China, 2002.

Part II
Graph Mining and Community Analysis

Chapter 4
A Survey of Link Mining Tasks for Analyzing Noisy and Incomplete Networks

Galileo Mark Namata, Hossam Sharara, and Lise Getoor

Abstract Many data sets of interest today are best described as networks or graphs of interlinked entities. Examples include Web and text collections, social networks and social media sites, information, transaction and communication networks, and all manner of scientific networks, including biological networks. Unfortunately, often the data collection and extraction process for gathering these network data sets is imprecise, noisy, and/or incomplete. In this chapter, we review a collection of link mining algorithms that are well suited to analyzing and making inferences about networks, especially in the case where the data is noisy or missing.

4.1 Introduction

A key emerging challenge for data mining is tackling the problem of mining richly structured, heterogeneous data sets. These kinds of data sets are best described as networks or graphs, where the nodes can be of different types, and the edges (or hyperedges) can represent different kinds of links. As evidenced by this volume, there has been a growing interest in methods which can mine and make inferences about such data (see also an earlier survey article and special issue issue of KDD Explorations [41]).

In the context of network data, statistical inference can be used in a variety of ways. Two of the most common are for inferring missing information and identifying (and correcting) incorrect network data. Furthermore, one way of understanding the different inference tasks in network data is according to whether they predict (or correct) information associated with nodes, edges, or larger subgraphs of the network. The inference task may be about inferring missing values (such as the label or attribute values for a node or edge), reasoning about the existence of nodes and edges (such as predicting whether two nodes should be merged because they refer to the

L. Getoor (✉)
Department of Computer Science, University of Maryland, College Park, MD, USA
e-mail: getoor@cs.umd.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_4,

same underlying entity, predicting whether a relationship exists), or reasoning about the existence of groupings of nodes and edges (group or community detection).

Examples of work applying statistical inference to infer missing or incorrect network data can be found in various domains. For example, in the social sciences, there is interest in studying human interaction from large online social networks [69, 113]. In these large online networks, individuals may own multiple accounts which need to be resolved to get an accurate count of the individuals in the network. Furthermore, the relationships (e.g., unspecified friends), attributes (e.g., gender), and membership in social groups (e.g., political affiliation) of the individuals of interest may not be given and need to be inferred. Similarly, in biology, there is interest in gaining new insight into biological processes by studying protein–protein interaction (PPI) networks [50, 107, 118]. The high-throughput methods typically used to create and annotate these networks are notoriously noisy and incomplete. Even the proteins of the most studied organisms, yeast, are not completely annotated with their functions and complex memberships and it is estimated that up to 52% the interactions for the current yeast PPI are spurious [50]. Analysis of these PPI networks requires applying statistical inference to infer the missing and correct function, interaction, and complex membership of proteins. As a final example, in computer networks, there is work in creating a map of the Internet to understand its vulnerabilities and limitations [105]. While some ISPs and research networks publish high-level topologies, in general the information about the topology and attributes of a large part of the Internet are privately owned and rarely published. Consequently, research in mapping the Internet mainly relies on inexact techniques which can only give a partial view of the global picture. Inference needs to be applied to the noisy and incomplete map to resolve IP addresses to routers and autonomous systems (AS), predict the existence and type of links between AS, and discover well-connected (and poorly connected) parts of the Internet.

All of the above examples require data mining and machine learning algorithms which can help to clean and improve the quality of the networks, before they are analyzed. In this chapter, we survey a subset of the inference tasks that are particularly useful in dealing with noisy and incomplete network data. We begin with some notation and then describe methods for *collective classification* (Section 4.3), *link prediction* (Section 4.4), *entity resolution* (Section 4.5), and *group detection* (Section 4.6).

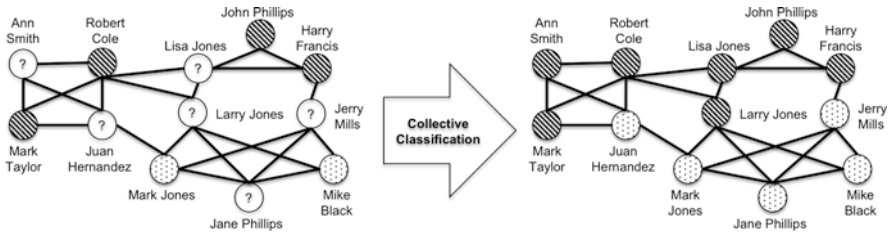


Fig. 4.1 Example of a collective classification problem. Nodes with a *question mark* are nodes whose labels are unknown. Collective classification uses the attributes and labels of neighboring nodes. Ann Smith, for example, is likely to have the same research area as her co-authors, Robert Cole and Mark Taylor

4.2 Terminology and Notation

We begin by introducing some general notation and terminology used through this chapter. First, let $G(V, E)$ denote a graph G with nodes $v \in V$ and edges $e \in E$. $|V|$ and $|E|$ are used to denote the size of the node and edge sets in the graph, respectively. We describe an edge and the nodes on that edge as *incident* to each other. Also, we refer to nodes which share an edge as *adjacent* to each other. For this document, whenever we use the term graph, we normally refer to either a directed graph (where each edge, $e \in E$, consists of an ordered pair of vertices) or undirected graph (where each edge, $e \in E$, consists of an unordered pair of vertices); in both cases, the edges are incident to exactly two nodes (i.e., $e = (v_i, v_j)$). In some cases, we refer to a *bipartite graph*, where the nodes can be divided into two disjoint sets, $V_1, V_2 \subset V$, so that every edge has one node in each of the two sets (i.e., $v_i \in V_i$, $v_j \in V_j$). Although we mainly use the terms graph, nodes, and edges in this chapter, we note that graphs are often used to represent networks, and the terms edges, link, and relationships are often used interchangeably.

Finally, throughout this chapter, we use a simple author collaboration network to illustrate the different inference tasks (shown in Figs. 4.1, 4.2, 4.3, and 4.4). In the collaboration network figures, the nodes represent authors and the edges between the authors indicate that the authors have co-authored at least one paper together. The shading of the nodes indicates the research area of the authors; to make it simple, here we assume there are just two areas, shown either in white (i.e., theory) or gray (i.e., systems), if observed, and as a "?" if it is unobserved. The bounded areas (as shown in Fig. 4.4) indicate group structure.

4.3 Collective Classification

A traditional problem in machine learning is to classify objects: given a corpus of documents classify each according to its topic label; given a collection of email communications determine which are not spam; given individuals in a collaboration network determine a characteristic of that individual; given a sentence, determine the part of speech for each word, etc. In networks, the problem of inferring labels has traditionally been applied to the nodes of the graph. Initial work in classification makes an independent and identically distributed (IID) assumption where the class label of each object is made in isolation. In graphs, however, studies have shown that predicting the labels of nodes can benefit by using autocorrelations between the node label and the attributes of related nodes. For example, in the collaboration network in Fig. 4.1, nodes with a question mark represent authors whose research areas are unknown. While we can use attributes of the author (e.g., titles of their papers) to predict the label, we can also use the research areas of the other authors they share a co-authorship edge with. The author, Ann Smith, for one is likely to work in theory given she has only co-authored with individuals in the theory field.

In the past decade there have been a number of approaches proposed which attempt to classify nodes in a joint or collective manner instead of treating each

in isolation. In the following sections, we formally define the problem of collective classification and introduce several types of approaches that have been proposed to address it.

4.3.1 Definition

Collective classification is an optimization problem where we are given the set of nodes, $V = \{v_1, v_2, ..., v_n\}$, over a graph $G(V, E)$, with a set of pre-defined labels, $L = \{l_1, l_2, ..., l_q\}$. Each node $v \in V$ can take exactly one value from the set of labels in L, denoted as $v.L$. Moreover, V is divided into two sets of nodes: V_k, the nodes for which we know the correct labels and V_u, the nodes whose labels need to be determined. We are also given a neighborhood function, N, over the nodes where $N_i \subseteq V \setminus v_i$, which captures the relationships of a node, v_i. The task of collective classification is to infer the values of the labels $v.L$ for the nodes $v \in V_u$.

4.3.2 Approaches

In this section, we describe the three main categories of collective classification algorithms which vary based on their mathematical underpinnings as well as how they exploit the relationships between the nodes.

4.3.2.1 Relational Classifiers

Traditional classification concentrates on classifying a given node using only the observed attributes of that node. Relational classifiers [104] go beyond that by also considering the observed attributes of related nodes. For instance, when classifying authors, not only would we use the words present in their papers, we would also look at the authors who they have co-authored with and their word usage and research area (if known) to arrive at the correct class label. One relational classifier, popular due to its simplicity, is the relational classifier proposed by Macskassy and Provost [73]. Their classifier makes two assumptions: some node labels are known and related nodes are likely to have the same labels. The classifier assigns a label to a node, v_i, by looking at the labels of related nodes whose label values are known, $N_i \cap V_u$, and taking the weighted proportion of neighbors for each possible label. The label with largest weighted proportion among neighbors is the predicted label of v_i. Although relational classifiers have been shown to perform well in some domains, overall the results have been mixed. For instance, although there have been reports of classification accuracy gains using such techniques over traditional classification, in certain cases, these techniques can harm classification accuracy [22].

4.3.2.2 Approaches Based on Local Conditional Classifiers

A source of information in collective classification is to use not only the attributes and the known labels of related nodes, but also the predicted labels of other nodes

whose labels are unobserved. For instance, going back to the classification example in Fig. 4.1, authors which share a co-authorship edge to other authors *predicted* to have a certain research area, are likely to work in the same area. In this section, we look at this source of information and exploit it using local conditional classifiers. Chakrabarti et al. [22] illustrated the use of this approach and reported impressive classification accuracy gains for labeling Web pages. Neville and Jensen [81] further developed the approach as an iterative classification algorithm (ICA) and studied the conditions under which it improved classification performance [57].

We provide pseudocode for a simple variant of ICA in Algorithm 1. The basic premise behind ICA is simple. Consider a node $v_i \in V$ whose label needs to be determined. Suppose we know the attributes and labels of related nodes, N_i, ICA assumes that we are given a local classifier f that takes the attributes and labels of the nodes in N_i and returns the most likely value of $v_i.L$. This makes the local classifier f an extremely flexible function and we can use popular classifiers like decision trees [95] and SVM [58] in its place. However, since N_i may contain nodes whose labels we also need to predict, we need to repeat the process iteratively where in each iteration, we label each $v_i.L$ using the current best estimates of N_i and classifier f. We continue to do so until the assignments to the labels stabilize or some stopping criterion is met.

Algorithm 1 Iterative Classification Algorithm

Iterative Classification Algorithm (ICA)

```
for each node v_i ∈ V do {bootstrapping}
    {c}ompute label using only observed nodes in N_i
    compute a_i using only V_k ∩ N_i
    v_i.L ← f(a_i)
end for
repeat {iterative classification}
    generate ordering O over nodes in V_u
    for each node v_i ∈ O do
        {c}ompute new estimate of v_i.L
        compute a_i using current assignments to N_i
        v_i.L ← f(a_i)
    end for
until all class labels have stabilized or a threshold number of iterations have elapsed
```

A number of aspects of the iterative approach have been studied. An important aspect is how to use the values provided by N_I in f [70]. Most classifiers are defined as functions with a fixed-length vector of attribute values as arguments while the number of nodes in N_i may vary for different v_i. A common approach to address this is to use an aggregation operator such as count, mode, or prop, which measures the proportion of neighbors with a given label. In Algorithm 1, we use $\mathbf{a}_i$ to denote the vector encoding the values in N_i obtained after aggregation. Another aspect to consider is the choice of the local classifier f. Classifiers used include naive Bayes [22, 81], logistic regression [70], decision trees [57], and weighted-vote [73].

There is some evidence to indicate that discriminately trained local classifiers such as logistic regression tend to produce higher accuracies than others [101].

Previous work has also looked at different ways of ordering and updating the labels in ICA. While there is some evidence which shows ICA is fairly robust to simple ordering strategies such as random ordering, visiting nodes in ascending order of diversity of its neighborhood class labels or labeling nodes in descending order of label confidence [40], strategies which vary what labels are updated at each iteration have been shown to improve accuracies [76].

Extensions have also been proposed for the ICA algorithm. Researchers in collective classification [73, 76, 82] have extended the simple algorithm described in Algorithm 1 and developed a version of Gibbs sampling that is easy to implement and faster than traditional Gibbs sampling approaches. The basic idea behind this algorithm is to assume, just like in the case of ICA, that we have access to a local classifier f that can sample for the best label estimate for $v_i.L$ given all the values for the nodes in N_i. We keep doing this repeatedly for a fixed number of iterations (a period known as burn-in). After that, not only do we sample for labels for each $v_i \in V_u$, but we also maintain count statistics as to how many times we sampled a give label for node v_i. After collecting a predefined number of such samples, we output the best label assignment for node v_i by choosing the label that was assigned the maximum number of times to v_i during the sampling.

4.3.2.3 Approaches Based on Global Formulations

In addition to the local conditional classifier approaches discussed in Section 4.3.2.2, another approach to collective classification is to represent the problem with a high-level global graphical model and then using the learning and inference techniques for the graphical modeling approach to arrive at the correct classification. Graphical models which have been used include both directed [43] and undirected [62, 109] models. While these techniques can use both the labels and attributes of related nodes, we note that these techniques tend to be less efficient and scalable than the iterative collective classification techniques.

A common way of defining such a global model is by using a pairwise Markov random field (pairwise MRF) [109]. Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ denote a random variable graph where $\mathcal{V}$ consists of the two types of random variables: the unobserved, $\mathcal{Y}$, which need to be assigned from a label set $\mathcal{L}$, and observed variables, $\mathcal{X}$, whose labels are known. Let Ψ denote a set of *clique potentials* which contain three distinct types of functions. First, for each $\mathcal{Y}_i \in \mathcal{E}$, $\psi_i \in \Psi$ is a mapping $\psi_i : L \rightarrow \mathbb{R} \geq 0$, where $\mathbb{R} \geq 0$ is the set of non-negative real numbers. Next, for each $(\mathcal{Y}_i, \mathcal{X}_j) \in \mathcal{E}$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : L \rightarrow \mathbb{R} \geq 0$. The last type of function is for each $(\mathcal{Y}_i, \mathcal{Y}_j) \in \mathcal{E}$, $\psi_{ij} \in \Psi$ is a mapping $\psi_{ij} : L \times L \rightarrow \mathbb{R} \geq 0$.

Let x denote the values assigned to all the observed variables in $\mathcal{G}$ and let x_i denote the value assigned to $\mathcal{X}_i$. Similarly, let y denote any assignment to all the unobserved variables in $\mathcal{G}$ and let y_i denote a value assigned to $\mathcal{Y}_i$. For brevity of

notation we will denote by ϕ_i the clique potential obtained by computing $\phi_i(y_i) = \psi(y_i) \prod_{(\mathcal{Y}_i, \mathcal{X}_j) \in \mathcal{E}} \psi ij(y_i)$. A pairwise MRF can then be defined as follows:

Definition 1 A pairwise Markov random field (pairwise MRF) is given by a pair $\langle \mathcal{G}, \Psi \rangle$ where $\mathcal{G}$ is a graph and Ψ is a set of clique potentials with ϕ_i and ψ_{ij} as defined above. Given an assignment y to all the unobserved variables $\mathcal{Y}$, the pairwise MRF is associated with the probability distribution:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} \prod_{Y_i \in Y} \phi_i(y_i) \prod_{(Y_i, Y_j) \in E} \psi_{ij}(y_i, y_j)$$

where $\mathbf{x}$ denotes the observed values of X and

$$Z(x) = \sum_{\mathbf{y}'} \prod_{Y_i \in Y} \phi_i(y_i') \prod_{(Y_i, Y_j) \in E} \psi_{ij}(y_i', y_j').$$

Given a pairwise MRF, it is conceptually simple to extract the best assignments to each unobserved variable in the network. For instance, we may adopt the criterion that the best label value for $\mathcal{Y}_i$ is simply the one corresponding to the highest marginal probability obtained by summing over all other variables from the probability distribution associated with the pairwise MRF. Computationally, however, this is difficult to achieve since computing one marginal probability requires summing over an exponentially large number of terms. Hence, approximate inference algorithms are typically employed, the two most common being loopy belief propagation (LBP) and mean-field relaxation labeling. A comparison of these two approaches are given in [90, 101].

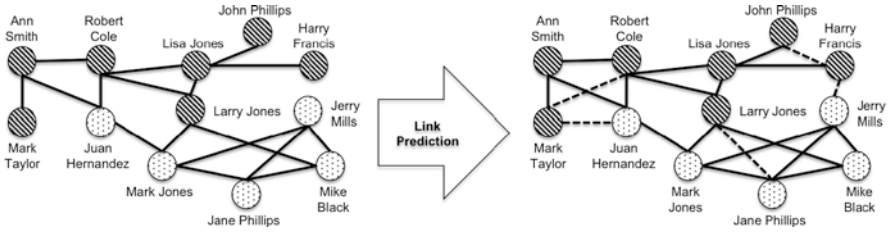


Fig. 4.2 Example of a link prediction problem. The graph on the *left* represents a collaboration network at time t, and the graph on the *right* represents the predicted collaboration network at time $t + 1$. Predicted collaboration edges are highlighted using a *dashed line*

4.4 Link Prediction

In this section, we change our focus from inferring information about the nodes of a network to inferring information about the links or edges between them. Inferring the existences of edges between nodes has traditionally been referred to as *link*

prediction [69, 110]. We provide a formal definition of the problem of link prediction, as well as discuss variants and closely related problems in Section 4.4.1.

Link prediction is a challenging problem that has been studied in various guises in different domains. For example, in social network analysis, there is work on predicting friendship links [119], event participation links (i.e., co-authorship [89]), communication links (i.e., email [89]), and links representing semantic relationships (i.e., advisor of [110] and subordinate manager [30]). In bioinformatics, there is interest in predicting the existence of edges representing physical protein–protein interactions [50, 107, 118], domain–domain interactions [29], and regulatory interactions [4]. Similarly, in computer network systems there is work in inferring unobserved connections between routers, and inferring relationships between autonomous systems and service providers [105]. There is also work on using link prediction to improve recommender systems [36, 51], Web site navigation [120], surveillance [52], and automatic document cross-referencing [77].

4.4.1 Definition

We begin with some basic definitions and notation. We refer to the set of possible edges in a graph as *potential edges*. The set of potential edges depends on the graph type and how the edges for the graph are defined. For example, in a directed graph, the set of potential edges consists of all edges $e = (v_1, v_2)$ where v_1 and v_2 are any two nodes V in the graph (i.e., the number of potential edges is $|V| \times |V|$). In an undirected bipartite graph with two subsets of nodes ($V_1, V_2 \in V$), while the edges still consist of a pair of nodes, $e = (v_1, v_2)$, there is an added condition such that one node must be from V_1 and the other node must be from V_2; this results in $|V_1| \times |V_2|$ potential edges. Next, we refer to set of "true" edges in a graph as *positive edges*, and we refer to the "true" non-edges in a graph (i.e., pairs of nodes without edges between them) as *negative edges*. For a given graph, typically we only have information about a subset of the edges; we refer to this set as the *observed* edges. The observed edges can include both positive and negative edges, though in many formulations there is an assumption of positive-only information. We can view link prediction as a probabilistic inference problem, where the evidence includes the observed edges, the attribute values of the nodes involved in the potential edge, and possibly other information about the network, and for any unobserved, potential edge, we want to compute the probability of it's existing. This can be reframed as a binary classification problem by choosing some probability threshold and concluding that potential edges with existence probability above the threshold are true edges, and those below the threshold are considered false edges (more complex schemes are possible as well).

The earliest and most cited formulation of the link prediction problem was proposed by Liben-Nowell and Kleinberg [69]. Liben-Nowell and Kleinberg [69] proposed a temporal formulation defined over a dynamic network where given a graph $G_t(V_t, E_t)$ at time t, infer the set of edges at the next time step $t + 1$. More formally,

the objective is to infer a set of edges E_{new} where $E_{t+1} = E_t \bigcup E_{\text{new}}$. In this chapter, we use a more general definition of link prediction proposed by Taskar et al. [110] where given a graph G and the set of potential edges in G, denoted $P(G)$, the problem of link prediction is to predict for all $p \in P(G)$ whether p exists or does not exists, remaining agnostic on whether G is a noisy graph with missing edges or a snapshot of a dynamic graph at a particular time point.

In addition to the definition of link prediction discussed above, it is also important to mention four closely related problems: *random graph models*, *link completion*, *leak detection*, and *anomalous link discovery*, whose objectives are different but very similar to link prediction. The first related research area, random graph models, is the problem of defining models for generating random graphs which capture the properties of graphs found in real networks [11, 33, 65, 66, 86]. Properties include scale-free degree distributions [1, 11, 35], the small-world phenomenon [11, 114], and densification and shrinking diameters of dynamic networks over time [66]. An important aspect of these models is modeling how to randomly generate edges between the nodes of the graph to capture these properties. The preferential attachment model [11], for example, creates edges based on the degree of nodes (i.e., higher degree nodes are more likely to be incident to more edges). The Forest Fire model [66], on the other hand, generates edges for nodes in an epidemic fashion, growing outward from some initial set of neighboring nodes.

The next two related problems, link completion [10, 21, 45] and leak detection [10, 20, 60], are a variation of link prediction over hypergraphs. A hypergraph is a graph where the edges (known as hyperedges) can connect any number of nodes. For example, in a hypergraph representing an email communication networks, a hyperedge may connect nodes representing email addresses that are recipients of a particular email communication. In link completion, given the set of nodes that participate in a particular hyperedge, the objective is to infer nodes that are missing. For our email communication network example, link completion may involve inferring which email address nodes need to be added to the hyperedge representing the recipients list of an email communication. Conversely, in leak detection, given the set of nodes participating in a particular hyperedge, the objective is to infer which nodes should not be part of that hyperedge. For example, in email communications, leak detection will attempt to infer which email address nodes are incorrectly part of the hyperedge representing the recipient list of the email communication.

The last problem, anomalous link discovery [53, 96], has been proposed as an alternate task to link prediction where the existence of the edges are assumed to be observed, and the objective is to infer which of the observed links are anomalous or unusual. Specifically, anomalous link discovery identifies which links are statistically improbable with the idea that these may be of interest for those analyzing the network. Rattigan and Jensen [96] show that some methods which perform poorly for link prediction can still perform well for anomalous link discovery.

4.4.2 Approach

In this section, we discuss the two general categories of the current link prediction models: topology-based approaches and node attribute-based approaches. Topology-based approaches are methods which rely solely on the topology of the network to infer edges. Node attribute-based approaches make predictions based on the attribute values of the nodes incident to the edges. In addition, there are models which make use of both structure and attribute values.

4.4.2.1 Topology-Based Approaches

A number of link prediction models have been proposed which rely solely on the topology of the network. These models typically rely on some notion of structural proximity, where nodes which are close are likely to share an edge (e.g., sharing common neighbors, nodes with a small shortest path distance between). The earliest topological approach for link prediction was proposed by [69]. In this work, Liben-Nowell and Kleinberg proposed various structure-based similarity scores and applied them over the unobserved edges of an undirected graph. They then use a threshold k and only predict edges with the top k scores as existing. A variety of similarity scores were proposed, given two nodes v_1 and v_2, including graph distance (the negated shortest path between v_1 and v_2), common neighbors (the size of the intersection of the sets of neighbors of v_1 and v_2), and more complex measures such as the Katz measure (the sum of the lengths of the paths between v_1 and v_2 exponentially damped by length to count short paths more heavily). Evaluating over a co-authorship network, the best performing proximity score measure was the Katz measure; however, the simple measures, which rely only on the intersection of the set of nodes adjacent to both nodes, performed surprisingly well. A related approach was proposed by [118] which applies the link prediction problem to predicting missing protein–protein interactions (PPI) from PPI networks generated by high-throughput methods. This work assumes that interacting proteins tend to form a clique. Thus, missing edges can be predicted by predicting the existence of edges which will create cliques in the network. More recent work by [24] has tried to go beyond predicting edges between neighboring nodes. In their problem domain of food webs, for example, pairs of predators often prey on a shared prey species but rarely prey on each other. Thus, in these networks, predicting "predator–prey" edges need to go beyond proximity. For this, they propose a "hierarchical random graph" approach which fits a hierarchical model to all possible dendrograms of a given network. The model is then used to calculate the likelihood of an edge existing in the network.

4.4.2.2 Node Attribute-Based Approaches

Although topology has been shown useful in link prediction, topology-based approaches ignore an important source of information in networks, the attributes of nodes. Often there are correlations in the attributes of nodes which share an

edge with each other. One approach which exploits this correlation was proposed by Taskar et al. [110]. In this approach, a relational Markov network (RMN) framework was applied to predicting the existence and class of edges between Web sites. They exploit the fact that certain links can only exist between nodes of the appropriate type. For example, an "advisor" edge can only exist between a student and a faculty nodes. Another approach which uses node attributes was proposed by [94]. In that approach, they used a structured logistic regression model over learned relational features to predict citation edges in a citation network. Their relational features are built over attributes such as the words used in the paper nodes. O'Madadhain et al. [89] also proposed an attribute based approach, constructing local conditional probability models based on the attributes such as node attribute similarity, topic distribution, and geographical location in predicting "co-participation" edges in an email communication network. More recently, there is work on exploiting other node attributes like the group membership of the nodes. Zheleva et al. [119] showed that membership in family groups are very useful in predicting friendship links in social networks. Similarly, [106] showed that using protein complex information can be useful in predicting protein–protein interactions. Finally, we note that in link prediction, as in classification, the quality of predictions can be improved by making the predictions collectively. Aside from the relational Markov network approach by [110] mentioned earlier, Markov Logic networks [98] and Probabilistic Relational models [42] have also been proposed for link prediction and are capable of performing joint inference.

4.4.3 Issues

There are a number of challenges which make link prediction very difficult. The most difficult challenge is the large class skew between the number of edges which exist and the number of edges which do not. To illustrate, consider directed graph denoted by $G(V, E)$. While the number of edges $|E|$ is often $O(|V|)$, the number of edges which do not exist is often $O(|V|^2)$. Consequently, the prior probability edge existence is very small. This causes many supervised models, which naively optimize for accuracy, to learn a trivial model which always predicts that a link does not exist. A related problem in link prediction is the large number of edges whose existence must be considered. The number of potential edges is $O(|V|^2)$ and this limits the size of the data sets which can be considered.

In practice, there are general approaches to addressing these issues either prior to or during the link prediction. With both large class skew and number of edges to contend with, the general approach is to make assumptions which reduce the number of edges to consider. One common way to do this is to partition the set of nodes where we only consider potential edges between nodes of the same partition; edges between partitions are not explicitly modeled and are assumed not to exist [2, 118]. This is useful in many domains where there is some sort of natural partition among the nodes available (e.g., geography in social networks, location of proteins in a

cell) which make edges across partitions unlikely. Another way is to define some simple, computationally inexpensive distance measure such that only edges whose nodes are within some distance are considered [30, 69].

Another practical issue in link prediction is that while real-world data often indicates which edges exist (positive examples), the edges which do not exist (negative examples) are rarely annotated for use by link prediction models. In bioinformatics, for example, the protein–protein interaction network of yeast, the most and annotated studied organism, is annotated with thousands of observed edges (physical interactions) between the nodes (proteins) gathered from numerous experiments [13]. There are currently, however, no major data sets available which indicate which proteins definitely do not physically interact. This is an issue not only in creating and learning models for link prediction but also an issue with evaluating them. Often, it is unclear whether a predicted edge which is not in our ground truth data is an incorrectly predicted edge or an edge resulting from incomplete data.

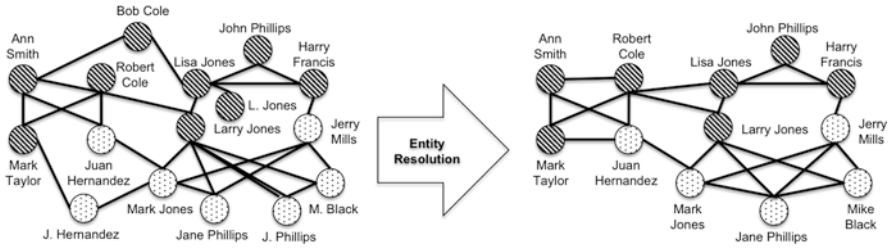


Fig. 4.3 Example of a entity resolution problem. In this example, the nodes on the *left* are ambiguous due to variations in the spelling of their names. While attributes may suffice to resolve the entities in some cases (e.g., Juan Hernandez and J. Hernandez are likely the same person due to the similarity in their names), some cases (e.g., J. Phillips can refer to either Jane or John Phillips) it may not. However, if we use the edges (i.e., both Jane Phillips and J. Phillips have collaborated with Larry Jones), we are able to improve our predictions

4.5 Entity Resolution

Many networks have uncertain and imprecise references to real-world entities. The absence of identifiers for the underlying entities often results in noisy networks which contain multiple references to the same underlying entity. In this section, we look at the problem of resolving which references refer to the same entity, a problem known as *entity resolution*.

Examples of entity resolution problems can be found in many domains, often under different names. The earliest applications of entity resolution is on medical data [37, 83, 84, 117]. In this work, in a problem they referred to as record linkage, the goal was to identify which medical records refer to the same individual or family. Later, in computer vision, entity resolution was applied in identifying which regions in the same image are part of the same object (the correspondence problem). Also,

in natural language processing, there is interest in determining which noun phrases refer to the same underlying entity (coreference resolution, object consolidation). The problems of deduplication and data integration, determining when two tuples in or across databases refer to the same entity, can also be seen as entity resolution.

4.5.1 Definition

We begin by introducing some additional notation. For a graph $G(V, E)$ we are given a set of reference nodes $R \subseteq V$ where the reference nodes correspond to some set of unknown entity nodes E. We introduce the notation $r.E$ to refer to the entity to which r corresponds. Formally, the general goal of entity resolution is to recover the hidden set of entities E and the entity labels $r.E$ for all the reference nodes.

We note that there are two commonly used interpretations of entity resolution and which is more natural depends on the algorithm chosen. First, entity resolution can be viewed as a pairwise classification problem, where for each pair of references, $r_i, r_j \in R$, we are interested in determining whether r_i and r_j are co-referent (i.e., $r_i.E = r_j.E$). Note the similarity here with link prediction; in fact, many of the challenges of link prediction (class skew and scaling) are issues in entity resolution as well. The second view is as a clustering problem, where the goal is to assign the reference nodes to clusters $C \in \mathcal{C}$. The subset of reference nodes in each cluster are assumed to be co-referent to each other (i.e., $\forall r_i, r_j \in C,\ r_i.E = r_j.E$).

4.5.2 Approach

In this section, we survey existing entity resolution approaches. We distinguish between three categories of approaches: attribute-based, naive relational, and collective relational. Attribute-based approaches are the traditional approaches to entity resolution which rely solely on the attributes of the reference nodes. More recently, naive and collective relational approaches have been proposed which take the edges between these nodes into consideration. The naive relational approaches consider the attribute similarity of related reference node. The collective relational approaches, on the other hand, use the edges to make decisions jointly.

4.5.2.1 Attribute-Based Entity Resolution

The attribute-based approach to entity resolution typically uses the pairwise formulation of the entity resolution problem [26, 37, 48]. Given two reference nodes, $r_i, r_j \in R$, the attribute-based approaches generally make use of a similarity measure, $sim_A(r_i, r_j)$, or a weighted combination of multiple similarity measures, over the attributes of the reference nodes. Several sophisticated similarity measures have been proposed for use in entity resolution based on the types of features and domain

knowledge. For example, there are string similarity measures used commonly over the names of an entity such as

- Jaccard [54]: the size of the intersection among the characters divided by the size of the union of the characters occurring.
- Jaro and Jaro-Winker [56, 117]: string similarity scores which attempt to take into account typical spelling deviation by looking at the similarity within a certain neighborhood of the string characters; the Jaro-Winkler score is based on Jaro and weights matches at the beginning more highly.
- Levenshtein (edit distance) [67]: the minimum number of insertions, deletions, and substitutions required to transform one string to the other.
- Monge-Elkan [78]: recursive subcomponent matching algorithm which looks at matching subcomponents of the strings; it is good at finding swapped fields, such as first and last names.

Approaches have also been proposed which learn a string similarity measure from labeled data [18]. Pairs of nodes whose similarity is above a certain threshold are predicted as co-referent. Transitivity may also be enforced such that if r_i and r_j are predicted co-referent and r_j and r_k predicted co-referent, r_i and r_k are also predicted co-referent.

4.5.2.2 Naive Relational Entity Resolution

While attribute-based approaches have been shown to do well in some domains, work in relational data has focused on incorporating links, in particular, co-occurrence information. The earliest work using links for entity resolution was explored in the database community. Ananthakrishna et al. [6] introduce a method for deduplication using edges in data warehouse applications where there is a dimensional hierarchy over the link relations. Kalashnikov et al. [59] proposed the Relationship-based Data Cleaning (RelDC) approach which uses graph theoretic techniques to discover and analyze relationships, such as affiliation and co-authorship, that exist between reference nodes.

4.5.2.3 Collective Relational Entity Resolution

Although the approaches in Section 4.5.2.2 consider the edges for entity resolution, only the attributes of linked references are considered and the different resolution decisions are still taken independently. Work in collective relational entity resolution addresses this by using the edges between nodes to establish dependencies in the resolution decisions. In databases, for example, approaches have been proposed [14, 32] where one resolution decision affects another if they are linked. Bhattacharya and Getoor [14, 17] propose different measures for edge similarity and show how those can be combined with attribute similarity iteratively to perform entity resolution on collaboration networks. Dong et al. [32] collectively resolve

entities of multiple types by propagating evidence along the links in a dependency graph. In machine learning, probabilistic models have also been proposed to consider the interactions between the different entity resolution decisions. McCallum and Wellner [75] use conditional random fields for noun coreference and use clique templates with tied parameters to capture repeated relational structure. Singla and Domingos [103] use the idea of merging evidence to allow the flow of reasoning between different pairwise decisions over multiple entity types. Markov logic networks have also been applied for collective entity resolution [93, 103]. Pasula et al. [92] propose a generic probabilistic relational model framework for performing entity resolution on citations. Li et al. [68] propose a probabilistic generative model which captures a joint distribution over pairs of entities in terms of co-mentions in documents. Similarly, Bhattacharya and Getoor [16] proposed a generative group model by extending the Latent Dirichlet Allocation model for documents and topics.

4.5.3 Issues

A major issue in entity resolution is that it is a known hard problem computationally; a naive algorithm is $O(N^2)$, which for very large data sets is not feasible. For many networks, it is infeasible to compare all pairs of references for approaches which use expensive similarity measures. Similarly, for many probabilistic models, it is infeasible to explicitly represent all the variables required for the inference. Thus, efficiencies have long been a focus for research in entity resolution. One mechanism for doing this involves computing the matches efficiently and employing techniques commonly called "blocking" to place nodes into disjoint "blocks" using cheap and index-based similarity computations [49, 79]. The number of potential pairs is greatly reduced by assuming that only pairs of nodes in the same block can be co-referent pairs. Another mechanism, proposed by McCallum et al. [74], relaxes the use of disjoint blocks and places nodes into possibly overlapping subsets called "canopies". Potential co-referent pairs are then restricted only to pairs of nodes which share at least one common canopy.

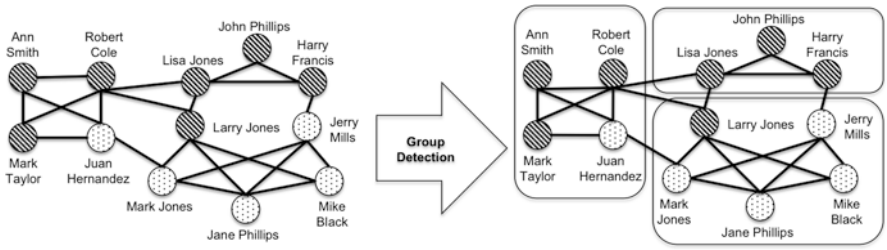


Fig. 4.4 Example of a group detection problem. The goal of group detection is to predict the underlying groups which the nodes, and/or edges, participate in. The three regions surrounded with a *rounded rectangle* represent the affiliations of our authors

Another issue in entity resolution is referred to a "canonicalization" [27, 116]. Once the reference nodes have been resolved to their corresponding entities, there is the problem of constructing a standard representation of the entity from the attributes of those references. In particular, canonicalization resolves the inconsistencies in the attributes among the reference nodes. Simple heuristics for determining the appropriate values for the attributes and edges of an entity based on the attributes of the references are possible; often these amount to choosing the longest string, or the most recently updated value. Such approaches, however, are not robust to noisy and incomplete attributes. Another approach is, instead of returning a single value for an attribute, keeping all the values, returning a ranked list of the possible values and edges [7, 111]. When there are a large number of references, however, the ranked list may be too long. Culotta et al. [27] addresses this by using adaptive similarity measures to select values in order to create a standard representation most similar to each of the different records. A unified approach was also proposed by Wick et al. [116] which performs entity resolution and canonicalization jointly using discriminatively trained model.

4.6 Group Detection

Another common problem that often occurs in reasoning about network data is inferring the underlying hidden groups or community structures of the nodes in the network. This problem is strongly related to data clustering; a traditional unsupervised learning problem in data mining. In cluster analysis, data points are organized in different groups based on the similarity of their feature values [55], where points in the same cluster are more similar to each other than points in different clusters according to a specific similarity measure. Similarly, a community in a network can be defined as a group of nodes that share dense connections among each other, while being less tightly connected to nodes in different communities in the network.

The importance of identifying the communities in networks lies in the fact that they can often be closely related to functional units of the system, e.g., groups of individuals interacting with each other in a society [8, 44, 71], WWW pages related to similar topics [38], compartments in food webs [61], or proteins responsible for similar biological functions [23]. Furthermore, analyzing the community structure itself provides insight into understanding the various roles of different nodes in their corresponding groups. For instance, by studying the structural properties of communities, one can distinguish between the functions of the central nodes in the group and the ones at the periphery.

In this section, we review some of basic methods for group detection and community discovery in network settings.

4.6.1 Definition

As before, we consider a graph $G = (V, E)$; in the case of weighted networks, $w(v_i, v_j)$ denotes the weight of the edge connecting nodes v_i and v_j. A community

or a group C is a subgraph $C(V', E')$ of the original graph $G(V, E)$ whose nodes and edges are subsets of the original graph's nodes and edges, i.e., $V' \subset V$ and $E' \subset E$. For each node v' in group C of G, we define an *internal* and an *external* degree as $d_{\text{int}}(v') = |e'(v', v_t)|; v_t \in V'$ and $d_{\text{ext}}(v') = |e'(v', v_t)|; v_t \notin V'$, where the internal degree of a node with respect to a certain group is the number of edges connecting it to other nodes of the group, while its external degree is the number of edges connecting it to nodes in the graph other than those in the corresponding group. Intuitively, nodes with relatively high internal degree and low external degree for a specific group are potentially good candidates to be included in that group. The opposite is also true, where nodes with low internal degree and high external degree for a specific group are candidates for removal. Throughout the discussion, the terms group, community, and cluster are used exchangeably.

To identify communities in networks, a basic set of properties that is capable of distinguishing a true community structure from a randomly selected set of nodes and edges is needed. One of the important properties that can be utilized is the graph density, which is the number of edges present in the network relative to the total number possible. Similarly, the density of a group of nodes in the network can be defined as the ratio between the number of edges connecting pairs of nodes within that group and the maximum number of possible edges within the same group:

$$\delta(C) = \frac{|E'|}{|V'| \times (|V'| - 1)/2}. \tag{4.1}$$

A randomly selected set of nodes from a network is likely to have a density similar to that of the global network structure. However, for community structures, the density of a group is expected to be higher than that of the overall graph. Formally, for any community C in a graph G, it is expected that $\delta(C) > \delta(G)$, where $\delta(G)$ is the overall graph density. Similarly, the average density of sets of nodes belonging to different communities, calculated using the ratio between the number of edges emanating from a group and terminating in another, and the maximum number possible of such edges, should generally be low. This basic idea is exploited in many of the group detection methods described next.

4.6.2 Approaches

Beyond the intuitive definition above, precisely defining what constitutes a community involves a number of aspects: whether the definition relies on global or local network properties, whether nodes can simultaneously belong to several communities, whether link weights are utilized, and whether the definition allows for hierarchical community structure. Global methods utilize the whole network structure for defining the communities. This can be achieved in several ways, such as global optimization methods [87, 97], algorithms based on different global centrality measures [39, 44], spectral methods [9, 31], or information-theoretic methods

[99, 100]. Local methods, on the other hand, define communities based on purely local network structure, such as detecting cliques of different sizes [34], clique percolation method [91], and subgraph fitness method [63].

As mentioned above, another important aspect is whether nodes are allowed to belong simultaneously to several communities. In general, overlapping communities do commonly occur in natural settings, especially in social networks. Currently, only a few methods are able to handle overlapping communities [88, 91]. Another difficulty in community detection is that networks may contain hierarchical structures, which means that communities may be parts of even larger communities. This leads to the problem of evaluating the best partitioning among different alternatives. One solution for evaluating the quality of a given community structure was suggested by Girvan and Newman [87], who introduced the concept of modularity as a measure for the goodness of a partitioning.

The methods used for community detection with respect to different perspectives are briefly reviewed in the following sections.

4.6.2.1 Clique-Finding Techniques

Cliques are graph structures that are frequently used in local techniques for community detection. A clique is defined as a complete subgraph $\{C(V', E') : \forall v_1, v_2 \in V', \exists(v_1, v_2) \in E'\}$, where there exists an edge between every pair of nodes belonging to it. In this context, communities can be considered as maximal clique, which cannot be extended with the addition of any new nodes or edges.

One of the problems of using this approach for group detection is the fact that finding cliques in a graph is an NP-complete problem. Another problem arises from the interpretation of communities, especially in social networks, where we expect different individuals to have different centrality in their corresponding groups, contradicting with the degree symmetry of nodes in cliques. To overcome these drawbacks, the notion of cliques is often relaxed to k-clique, which is a maximal subgraph where the distance between each pair of its nodes is not larger than k [3].

Recently, Palla et al. [91] introduced a local method for community detection called the clique percolation method. The method is based on the observation that, due to the high density of community structures, it is more likely that nodes within a given community form more small-sized cliques than nodes belonging to different communities. The clique percolation algorithm defines communities by considering overlapping chains of small cliques, which are likely to explore a significant fraction of each community, without crossing the boundary between different communities. Specifically, a community of size k is obtained by "rolling" a clique of size k over cliques of the same size that share at least $k - 1$ nodes with the current clique.

4.6.2.2 Clustering Techniques

Data clustering is one of the earliest techniques for group detection, where data points are grouped according to a specific similarity measure over their features.

The main objective of traditional clustering methods is to obtain clusters or groups of data points possessing high intra-cluster similarity and low inter-cluster similarity. Classical data clustering techniques can be divided into partition-based methods such as *k*-means clustering [72], model-based methods such as *Expectation-Maximization* algorithm [28], spectral clustering algorithms [5, 115] and hierarchical clustering methods [47] which are very popular and commonly used in many fields.

One advantage of the hierarchical clustering techniques is that they provide the ability to look at the groups at multiple resolutions. Hierarchical techniques are further divided into agglomerative and divisive algorithms. The agglomerative algorithm is a greedy bottom-up algorithm which starts with individual data points, then successively merge pairs with highest similarity. At each iteration, the similarities between the new cluster and each of the old clusters are recomputed and again the maximally similar pair of clusters merged. Divisive algorithms work in a reverse manner, where initially the whole set of points is regarded as one cluster which is successively divided into smaller ones by splitting nodes of lowest similarity. In both algorithms, clusters are represented as a dendrogram (see Fig. 4.5), whose depths indicate the steps at which two clusters are joined. This representation provides insight into the formed groups, where it is clear which communities are built up from smaller modules, and how these smaller communities are organized.

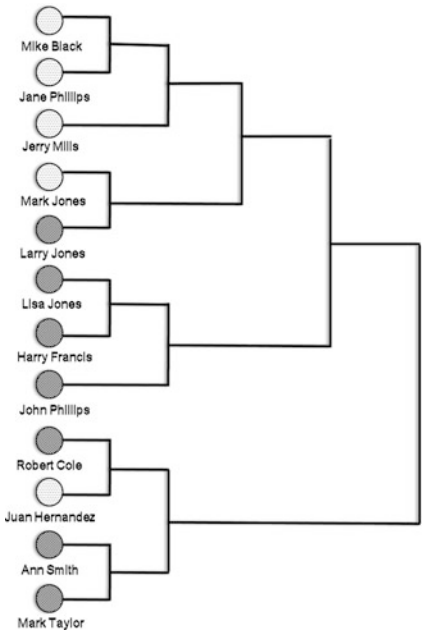


Fig. 4.5 A dendrogram resulting from a hierarchical clustering technique. Different levels in the tree correspond to partitions of the graph into clusters

Hierarchical clustering techniques can easily be adapted to network domains, where data points are replaced by individual nodes in the network, and the similarity is based on edges between them. In addition, there are other divisive algorithms based on spectral methods and other community detection techniques, which are discussed in the following sections.

4.6.2.3 Centrality-Based Techniques

Girvan and Newman introduced several community detection algorithms that have received much attention. The first method [44] uses a divisive algorithm based on the betweenness centrality of edges to be able to recover the group structure within the network. Betweenness centrality is a measure of centrality of nodes in networks, defined for each node as the number of shortest paths between pairs of nodes in the network that run through it. The Girvan–Newman algorithm extended this definition for edges in the network as well, where the betweenness centrality of an edge is defined as the number of shortest paths between pairs of nodes that include this edge.

The algorithm is also based on the fact that there exists denser connections between nodes belonging to the same group structure than those in different groups. Thus, all shortest paths between nodes from different communities should pass along one of these sparse set of edges, increasing their edge betweenness centrality measure. By following a divisive approach and removing edges with highest betweenness centrality from the network successively, the underlying community structure is revealed.

One of the drawbacks of the algorithm is its time complexity which is $O(|E|^2|V|)$ generally, and $O(|V|^3)$ for sparse networks. However, by limiting the re-calculations of the edge betweenness for only those affected by the prior edge removal can be factored in, making the algorithm efficient in sparse networks with strong community structure, but still not very efficient on dense networks. Following the same approach, other methods based on different notions of centrality have been introduced [64, 112].

4.6.2.4 Modularity-Based Techniques

The concept of modularity was introduced by Newman and Girvan [87] as a measure to evaluate the quality of a set of extracted communities in a network and has become one of the most popular quality functions used for community detection. The basic idea is utilizing a *null model*; a randomly rewired version of the original network preserving the node degrees, which is expected to contain no community structure. Modularity is then calculated by comparing the number of edges within the extracted communities against the expected number of edges in the same communities from the random network. More specifically, the modularity Q is defined as follows:

$$Q = \frac{1}{2|E|} \sum_{ij} \left[A_{ij} - \frac{k_i . k_j}{2|E|} \right] \delta(c_i, c_j), \tag{4.2}$$

where A_{ij} is the element of the adjacency matrix of the network denoting the number of edges between nodes i and j, k_i and k_j are the degrees of nodes i and j respectively, c_i and c_j are the communities to which nodes i and j belong respectively. The summation runs over all pairs of nodes within the same community.

Clearly, a higher modularity value indicates that the average density of the extracted community is larger than that of the random network where no community structure is present. Thus, modularity maximization can be used as the objective for producing high-quality community structure. However, modularity maximization is an NP-hard problem [19]. Nevertheless, there has been several heuristics for approximate modularity maximization with reasonable time complexity.

An efficient greedy modularity maximization algorithm was introduced by Newman [85]. The algorithm starts with individual nodes and merges them agglomeratively, by choosing the pair that gives the largest increase in modularity. The time complexity of this greedy algorithm is $O(|V|(|E| + |V|))$ or $O(|V|^2)$ for sparse networks, which enables users to run community detection on large networks in a reasonable amount of time. A further speedup was achieved by Clauset et al. [25] by utilizing specialized data structures for sparse matrices.

4.6.3 Issues

Because the majority of work on group detection in relational setting has focused on the structural properties of the nodes and the edges in the underlying network, the resulting communities often lack a correspondence with the actual functional communities in the network [102]. Recently, relational clustering methods have been introduced for combining structural information with node characteristics to obtain better communities that are more related to the functional units in the network [15, 80]. However, more work is needed for tying the information about the target function with the group detection process to obtain different community structures from the network according to the specific function that needs to be highlighted.

One of the issues that has attracted more attention lately is the fact that most group detection methods works on single-mode networks, with less work focused on finding groups in more complex, multi-mode settings [12, 46]. Most algorithms deal with these types of networks by projecting them onto a series of individual graphs for each mode, thus losing some of the information that could have been retained by operating collectively on the original multi-modal setting.

Another issue that is gaining more interest is developing new methods for group detection in dynamic network settings [108], where the underlying network structure changes over time. Most of the previous work on group detection mainly focused on static networks, and handles the dynamic case by either analyzing a snapshot of the network at a single point in time, or aggregating all interactions over

the whole time period. Both approaches do not capture the dynamics of change in the network structure, which can be an important factor in revealing the underlying communities.

4.7 Conclusion

In this chapter, we have surveyed some of the common inference tasks that can be applied to graph data. The algorithms we have presented are especially well suited to the situation where we have noisy and incomplete observations. Some of the methods focus on predicting attribute values, some focus on inferring the existence of edges, and some focus on grouping nodes, either for entity resolution or for community detection. There are many other possibilities and combinations still to be explored, and this research area is likely to expand as we gather more and more graph and network data from a wider variety of sources.

Acknowledgments The work was supported by NSF Grant #0746930.

References

1. J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. In *Proceedings of the 6th Annual European Symposium on Algorithms*, Venice, Italy, 1998.
2. S. F. Adafre and M. de Rijke. Discovering missing links in wikipedia. In *Proceedings of the 3rd International Workshop on Link Discovery*, Chicago, IL, 2005.
3. R. D. Alba. A graph-theoretic definition of a sociometric clique. *Journal of Mathematical Sociology*, 3:113–126, 1973.
4. R. Albert, B. DasGupta, R. Dondi, S. Kachalo, E. Sontag, A. Zelikovsky, and K. Westbrook. A novel method for signal transduction network inference from indirect experimental evidence. *Journal of Computational Biology*, 14:407–419, 2007.
5. C. Alpert, A. Kahng, and S. Yao. Spectral partitioning: The more eigenvectors, the better. *Discrete Applied Math*, 90:3–26, 1999.
6. R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the 28th International Conference on Very Large Databases*, Hong Kong, China, 2002.
7. P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *Proceedings of the 22nd International Conference on Data Engineering*, Hong Kong, China, 2006.
8. A. Arenas, L. Danon, A. Daz-Guilera, P. M. Gleiser, and R. Guimer. Community analysis in social networks. *The European Physical Journal B*, 38(2):373–380, 2004.
9. A. Arenas, A. Daz-Guilera, and C. J. Prez-Vicente. Synchronization reveals topological scales in complex networks. *Physical Review Letters*, 96(11):114102, 2006.
10. R. Balasubramanyan, V. R. Carvalho, and W. Cohen. Cutonce- recipient recommendation and leak detection in action. In *Workshop on Enhanced Messaging*, Chicago, IL, 2009.
11. A.-L. Barabasi and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.
12. J. Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76:066102, 2007.

13. A. Ben-Hur and W. Noble. Choosing negative examples for the prediction of protein-protein interactions. *BMC Bioinformatics*, 7:S2, 2006.
14. I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *Data Mining and Knowledge Discovery*, Paris, France, 2004.
15. I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. In *ACM SIGKDD Workshop on Multi Relational Data Mining*, Chicago, Illinois, 2005.
16. I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *SIAM Conference on Data Mining*, Bethesda, MD 2006.
17. I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1:1–36, 2007.
18. M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., 2003.
19. U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. N. Z, and D. Wagner. On finding graph clusterings with maximum modularity. In *Proceedings of 33rd International Workshop on Graph-Theoretical Concepts in Computer Science*, Dornburg, Germany, 2007.
20. V. R. Carvalho and W. W. Cohen. Preventing information leaks in email. In *SIAM Conference on Data Mining*, Minneapolis, MN, 2007.
21. P. Chaiwanarom and C. Lursinsap. Link completion using prediction by partial matching. In *International Symposium on Communications and Information Technologies*, Vientiane, Lao, 2008.
22. S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *ACM SIGMOD International Conference on Management of Data*, Seattle, WA, 1998.
23. J. Chen and B. Yuan. Detecting functional modules in the yeast protein-protein interaction network. *Bioinformatics*, 22(18):2283–2290, 2006.
24. A. Clauset, C. Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98, 2008.
25. A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review*, 70(6):066111, 2004.
26. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the International Joint Conference on Artificial Intelligence Workshop on Information Integration*, Acapulco, Mexico, 2003.
27. A. Culotta, M. Wick, R. Hall, M. Marzilli, and A. McCallum. Canonicalization of database records using adaptive similarity measures. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose, CA, 2007.
28. A. P. Dempster, N. M. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B*, 39(1):1 – 38, 1977.
29. M. Deng, S. Mehta, F. Sun, and T. Chen. Inferring domain-domain interactions from protein-protein interactions. *Genome Research*, 12(10):1540–1548, October 2002.
30. C. Diehl, G. M. Namata, and L. Getoor. Relationship identification for social network discovery. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, Vancouver, Canada, 2007.
31. L. Donetti and M. A. Muoz. Detecting network communities: A new systematic and efficient algorithm. *Journal of Statistical Mechanics*, 10:10012, 2004.
32. X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Baltimore, MD, 2005.
33. P. Erdos and A. Renyi. On the evolution of random graphs. *Mathematics Institute Hungarian Academy of Science*, 5:17–61, 1960.
34. M. G. Everett and S. P. Borgatti. Analyzing clique overlap. *Connections*, 21(1):49–61, 1998.
35. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Cambridge, MA, 1999.
36. S. Farrell, C. Campbell, and S. Myagmar. Relescope: an experiment in accelerating relationships. In *Extended Abstracts on Human Factors in Computing Systems*, 2005.

37. I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
38. G. W. Flake, S. Lawrence, C. L. Giles, and F. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35:66–71, 2002.
39. S. Fortunato, V. Latora, and M. Marchiori. Method to find community structures based on information centrality. *Physical Review E*, 70(5):056104, 2004.
40. L. Getoor. *Advanced Methods for Knowledge Discovery from Complex Data*, chapter Link-based classification. Springer, London, 2005.
41. L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explorations Newsletter*, 7:3–12, 2005.
42. L. Getoor, N. Friedman, D. Koller, and B. Taskar. Learning probabilistic models of link structure. *Machine Learning*, 3:679–707, 2003.
43. L. Getoor, E. Segal, B. Taskar, and D. Koller. Probabilistic models of text and link structure for hypertext classification. In *International Joint Conferences on Artificial Intelligence Workshop on Text Learning: Beyond Supervision*, 2001.
44. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. In *Proceedings of National Academy of Science*, 2002.
45. A. Goldenberg, J. Kubica, P. Komarek, A. Moore, and J. Schneider. A comparison of statistical and machine learning algorithms on the task of link completion. In *Conference on Knowledge Discovery and Data Mining, Workshop on Link Analysis for Detecting Complex Behavior*, Washington, D.C., 2003.
46. R. Guimera, M. Sales-Pardo, and L. A. N. Amaral. Module identification in bipartite and directed networks. *Physical Review E*, 76:036102, 2007.
47. J. A. Hartigan. *Clustering Algorithms*. Wiley, New York NY, 1975.
48. O. Hassanzadeh, M. Sadoghi, and R. J. Miller. Accuracy of approximate string joins using grams. In *5th International Workshop on Quality in Databases at VLDB*, Vienna, Austria, 2007.
49. M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proc. of the ACM Sigmod International Conference on Management of Data*, San Jose, CA, 1995.
50. H. Huang and J. S. Bader. Precision and recall estimates for two-hybrid screens. *Bioinformatics*, 25(3):372–378, 2009.
51. Z. Huang, X. Li, and H. Chen. Link prediction approach to collaborative filtering. In *ACM/IEEE-CS Joint Conference on Digital Libraries*, 2005.
52. Z. Huang and D. K. J. Lin. The Time-Series Link Prediction Problem with Applications in Communication Surveillance. *Informs Journal On Computing*, 21:286–303, 2008.
53. Z. Huang and D. D. Zeng. A link prediction approach to anomalous email detection. In *IEEE International Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, 2006.
54. P. Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.
55. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.
56. M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14:491–498, 1995.
57. D. Jensen, J. Neville, and B. Gallagher. Why collective inference improves relational classification. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, WA, 2004.
58. T. Joachims. *Learning to Classify Text Using Support Vector Machines*. PhD thesis, University of Dortmund, 2002.
59. D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM International Conference on Data Mining*, Newport Beach, CA, 2005.
60. C. Kalyan and K. Chandrasekaran. Information leak detection in financial e-mails using mail pattern analysis under partial information. In *Proceedings of the 7th Conference on WSEAS*

International Conference on Applied Informatics and Communications, Athens, Greece, 2007.
61. A. E. Krause, K. A. Frank, D. M. Mason, R. E. Ulanowicz, and W. W. Taylor. Compartments revealed in food-web structure. *Nature*, 426(6964):282–285, 2003.
62. J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, Williamstown, MA, 2001.
63. A. Lancichinetti, S. Fortunato, and J. Kertesz. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11:033015, 2009.
64. V. Latora and M. Marchiori. Efficient behavior of small-world networks. *Physical Review Letters*, 87(19):198701, 2001.
65. J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, Nevada, 2008.
66. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1):2, 2007.
67. V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.
68. X. Li, P. Morie, and D. Roth. Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine Special Issue on Semantic Integration*, 26(1):45–58, 2005.
69. D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *International Conference on Information and Knowledge Management*, New Orleans, LA, 2003.
70. Q. Lu and L. Getoor. Link-based classification. In *Proceedings of the International Conference on Machine Learning*, 2003.
71. D. Lusseau and M. E. J. Newman. Identifying the role that animals play in their social networks. In *Proceedings of the Royal Society of London*, 2004.
72. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.
73. S. A. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study. *Journal of Machine Learning Research*, 8:935–983, 2007.
74. A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the 6th International Conference On Knowledge Discovery and Data Mining*, Boston, MA, 2000.
75. A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *International Workshop on Information Integration on the Web*, 2003.
76. L. McDowell, K. M. Gupta, and D. W. Aha. Cautious inference in collective classification. In *Association for the Advancement of Artificial Intelligence*, 2007.
77. D. Milne and I. H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and Knowledge Management*, Napa Valley, CA, 2008.
78. A. E. Monge and C. P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, 1996.
79. A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the Special Interest Group on Management of Data Workshop on Research Issues on Data Mining and Knowledge Discovery*, Tucson, AZ, 1997.
80. J. Neville, M. Adler, and D. Jensen. Clustering relational data using attribute and link information. In *Proceedings of the Text Mining and Link Analysis Workshop, 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.
81. J. Neville and D. Jensen. Iterative classification in relational data. In *Association for the Advancement of Artificial Intelligence Workshop on Learning Statistical Models from Relational Data*, 2000.

82. J. Neville and D. Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692, 2007.
83. H. B. Newcombe and J. M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Communications ACM*, 5(11):563–566, 1962.
84. H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, October 1959.
85. M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004.
86. M. E. J. Newman, A. L. Barabasi, and D. J. Watts. *The Structure and Dynamics of Networks*. Princeton University Press, Princeton, NJ, 2006.
87. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004.
88. M. E. J. Newman and E. A. Leicht. Mixture models and exploratory analysis in networks. In *Proceedings of National Academy of Science*, 2007.
89. J. O'Madadhain, J. Hutchins, and P. Smyth. Prediction and ranking algorithms for event-based network data. *SIGKDD Explorations Newsletter*, 7(2):23–30, 2005.
90. M. Opper and D. Saad, editors. *Advanced Mean Field Methods*. Neural Information Processing Series. MIT Press, Cambridge, MA, 2001. Theory and practice, Papers from the workshop held at Aston University, Birmingham, 1999, A Bradford Book.
91. G. Palla, I. Dernyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.
92. H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *Neural Information Processing Systems*, Vancouver, Canada, 2003.
93. H. Poon and P. Domingos. Joint unsupervised coreference resolution with markov logic. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Honolulu, HI, 2008.
94. A. Popescul and L. H. Ungar. Statistical relational learning for link prediction. In *International Joint Conferences on Artificial Intelligence Workshop on Learning Statistical Models from Relational Data*, Acapulco, Mexico, 2003.
95. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, USA, 1993.
96. M. J. Rattigan and D. Jensen. The case for anomalous link discovery. *SIGKDD Explorations Newsletter*, 7:41–47, 2005.
97. J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1):016110, 2006.
98. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.
99. M. Rosvall and C. T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. In *Proceedings of National Academy of Science*, 2007.
100. M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. In *Proceedings of National Academy of Science*, 2008.
101. P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.
102. C. R. Shalizi, M. F. Camperi, and K. L. Klinkner. Discovering functional communities in dynamical networks. *Statistical Network Analysis: Models, Issues, and New Directions*, pages 140–157, 2007.
103. P. Singla and P. Domingos. Entity resolution with markov logic. *IEEE International Conference on Data Mining*, 21:572–582, Hong Kong, China, 2006.
104. S. Slattery and M. Craven. Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of the 8th international Conference on Inductive Logic Programming*, Madison, Wisconsin, 1998.
105. N. Spring, D. Wetherall, and T. Anderson. Reverse engineering the internet. *SIGCOMM Computer Communication Review*, 34(1):3–8, 2004.

106. E. Sprinzak, Y. Altuvia, and H. Margalit. Characterization and prediction of protein-protein interactions within and between complexes. *Proceedings of the National Academy of Sciences*, 103(40):14718–14723, 2006.
107. A. Szilagyi, V. Grimm, A. K. Arakaki, and J. Skolnick. Prediction of physical protein-protein interactions. *Physical Biology*, 2(2):S1–S16, 2005.
108. C. Tantipathananandh and T. Y. Berger-Wolf. Algorithms for identifying dynamic communities. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France, 2009.
109. B. Taskar, A. Pieter, and D. Koller. Discriminative probabilistic models for relational data. In *Conference on Uncertainty in Artificial Intelligence*, Alberta, Canada, 2002.
110. B. Taskar, M.-F. Wong, P. Abbeel, and D. Koller. Link prediction in relational data. In *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2003.
111. S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 26:2001, 2001.
112. I. Vragovic and E. Louis. Network community structure and loop coefficient method. *Physical Review E*, 74(1):016105, 2006.
113. S. Wasserman, K. Faust, and D. Iacobucci. *Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, Cambridge November 1994.
114. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.
115. Y. Weiss. Segmentation using eigenvectors: A unifying view. In *Proceedings of International Conference on Computer Vision*, 1999.
116. M. L. Wick, K. Rohanimanesh, K. Schultz, and A. McCallum. A unified approach for schema matching, coreference and canonicalization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, Nevada, 2008.
117. W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.
118. H. Yu, A. Paccanaro, V. Trifonov, and M. Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006.
119. E. Zheleva, L. Getoor, J. Golbeck, and U. Kuter. Using friendship ties and family circles for link prediction. In *2nd ACM SIGKDD Workshop on Social Network Mining and Analysis*, Las Vegas, Nevada, 2008.
120. J. Zhu. *Mining Web Site Link Structure for Adaptive Web Site Navigation and Search*. PhD thesis, University of Ulster at Jordanstown, UK, 2003.

Chapter 5
Markov Logic: A Language and Algorithms for Link Mining

Pedro Domingos, Daniel Lowd, Stanley Kok, Aniruddh Nath, Hoifung Poon, Matthew Richardson, and Parag Singla

Abstract Link mining problems are characterized by high complexity (since linked objects are not statistically independent) and uncertainty (since data is noisy and incomplete). Thus they necessitate a modeling language that is both probabilistic and relational. Markov logic provides this by attaching weights to formulas in first-order logic and viewing them as templates for features of Markov networks. Many link mining problems can be elegantly formulated and efficiently solved using Markov logic. Inference algorithms for Markov logic draw on ideas from satisfiability testing, Markov chain Monte Carlo, belief propagation, and resolution. Learning algorithms are based on convex optimization, pseudo-likelihood, and inductive logic programming. Markov logic has been used successfully in a wide variety of link mining applications and is the basis of the open-source Alchemy system.

5.1 Introduction

Most objects and entities in the world are not independent, but are instead linked to many other objects through a diverse set of relationships: people have friends, family, and coworkers; scientific papers have authors, venues, and references to other papers; Web pages link to other Web pages and have hierarchical structure; proteins have locations and functions, and interact with other proteins. In these examples, as in many others, the context provided by these relationships is essential for understanding the entities themselves. Furthermore, the relationships are often worthy of analysis in their own right. In link mining, the connections among objects are explicitly modeled to improve performance in tasks such as classification, clustering, and ranking, as well as enabling new applications, such as link prediction.

P. Domingos (✉)
Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-2350, USA
e-mail: pedrod@cs.washington.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_5,

As link mining grows in popularity, the number of link mining problems and approaches continues to multiply. Rather than solving each problem and developing each technique in isolation, we need a common representation language for link mining. Such a language would serve as an interface layer between link mining applications and the algorithms used to solve them, much as the Internet serves as an interface layer for networking, relational models serve as an interface layer for databases, etc. This would both unify many approaches and lower the barrier of entry to new researchers and practitioners.

At a minimum, a formal language for link mining must be (a) relational and (b) probabilistic. Link mining problems are clearly relational, since each link among objects can be viewed as a relation. First-order logic is a powerful and flexible way to represent relational knowledge. Important concepts such as transitivity (e.g., "My friend's friend is also my friend"), homophily (e.g., "Friends have similar smoking habits"), and symmetry (e.g., "Friendship is mutual") can be expressed as short formulas in first-order logic. It is also possible to represent much more complex, domain-specific rules, such as "Each graduate student coauthors at least one publication with his or her advisor."

Most link mining problems have a great deal of uncertainty as well. Link data is typically very noisy and incomplete. Even with a perfect model, few questions can be answered with certainty due to limited evidence and inherently stochastic domains. The standard language for modeling uncertainty is probability. In particular, probabilistic graphical models have proven an effective tool in solving a wide variety of problems in data mining and machine learning.

Since link mining problems are both relational and uncertain, they require methods that combine logic and probability. Neither one alone suffices: first-order logic is too brittle and does not handle uncertainty; standard graphical models assume that data points are i.i.d. (independent and identically distributed), and do not handle the relational dependencies and variable-size networks present in link mining problems.

Markov logic [7] is a simple yet powerful generalization of probabilistic graphical models and first-order logic, making it ideally suited for link mining. A *Markov logic network* is a set of weighted first-order formulas, viewed as templates for constructing Markov networks. This yields a well-defined probability distribution in which worlds are more likely when they satisfy a higher-weight set of ground formulas. Intuitively, the magnitude of the weight corresponds to the relative strength of its formula; in the infinite-weight limit, Markov logic reduces to first-order logic. Weights can be set by hand or learned automatically from data. Algorithms for learning or revising formulas from data have also been developed. Inference algorithms for Markov logic combine ideas from probabilistic and logical inference, including Markov chain Monte Carlo, belief propagation, satisfiability, and resolution.

Markov logic has already been used to efficiently develop state-of-the-art models for many link mining problems, including collective classification, link-based clustering, record linkage, and link prediction, in application areas such as the Web, social networks, molecular biology, and information extraction. Markov logic makes

link mining easier by offering a simple framework for representing well-defined probability distributions over uncertain, relational data. Many existing approaches can be described by a few weighted formulas, and multiple approaches can be combined by including all of the relevant formulas. Many algorithms, as well as sample data sets and applications, are available in the open-source Alchemy system [17] (alchemy.cs.washington.edu).

In this chapter, we describe Markov logic and its algorithms and show how they can be used as a general framework for link mining. We begin with background on first-order logic and Markov networks. We then define Markov logic and a few of its basic extensions. Next, we discuss a number of inference and learning algorithms. Finally, we show two link mining applications, each of which can be written in just a few formulas and solved using the previous algorithms.

5.2 First-Order Logic

A *first-order knowledge base* (KB) is a set of sentences or formulas in first-order logic [9]. Formulas are constructed using four types of symbols: constants, variables, functions, and predicates. Constant symbols represent objects in the domain of interest (e.g., people: `Anna`, `Bob`, `Chris`). Variable symbols range over the objects in the domain. Function symbols (e.g., `MotherOf`) represent mappings from tuples of objects to objects. Predicate symbols represent relations among objects in the domain (e.g., `Friends`) or attributes of objects (e.g., `Smokes`). An *interpretation* specifies which objects, functions, and relations in the domain are represented by which symbols. Variables and constants may be *typed*, in which case variables range only over objects of the corresponding type, and constants can only represent objects of the corresponding type. For example, the variable x might range over people (e.g., Anna, Bob), and the constant `C` might represent a city (e.g, Seattle, Tokyo).

A *term* is any expression representing an object in the domain. It can be a constant, a variable, or a function applied to a tuple of terms. For example, `Anna`, x, and `GreatestCommonDivisor`(x, y) are terms. An *atomic formula* or *atom* is a predicate symbol applied to a tuple of terms (e.g., `Friends(x, MotherOf(Anna))`). Formulas are recursively constructed from atomic formulas using logical connectives and quantifiers. If F_1 and F_2 are formulas, the following are also formulas: $\neg F_1$ (negation), which is true iff F_1 is false; $F_1 \wedge F_2$ (conjunction), which is true iff both F_1 and F_2 are true; $F_1 \vee F_2$ (disjunction), which is true iff F_1 or F_2 is true; $F_1 \Rightarrow F_2$ (implication), which is true iff F_1 is false or F_2 is true; $F_1 \Leftrightarrow F_2$ (equivalence), which is true iff F_1 and F_2 have the same truth value; $\forall \mathtt{x}\ F_1$ (universal quantification), which is true iff F_1 is true for every object x in the domain; and $\exists x\ F_1$ (existential quantification), which is true iff F_1 is true for at least one object x in the domain. Parentheses may be used to enforce precedence. A *positive literal* is an atomic formula; a *negative literal* is a negated atomic formula. The formulas in a KB are implicitly conjoined, and thus a KB can be viewed as a single large formula. A *ground term* is a term containing no variables. A *ground atom* or *ground*

predicate is an atomic formula all of whose arguments are ground terms. A *possible world* (along with an interpretation) assigns a truth value to each possible ground atom.

A formula is *satisfiable* iff there exists at least one world in which it is true. The basic inference problem in first-order logic is to determine whether a knowledge base *KB entails* a formula F, i.e., if F is true in all worlds where *KB* is true (denoted by $KB \models F$). This is often done by *refutation*: *KB* entails F iff $KB \cup \neg F$ is unsatisfiable. (Thus, if a *KB* contains a contradiction, all formulas trivially follow from it, which makes painstaking knowledge engineering a necessity.) For automated inference, it is often convenient to convert formulas to a more regular form, typically *clausal form* (also known as *conjunctive normal form (CNF)*). A *KB* in clausal form is a conjunction of *clauses*, a clause being a disjunction of literals. Every *KB* in first-order logic can be converted to clausal form using a mechanical sequence of steps[1]. Clausal form is used in resolution, a sound and refutation-complete inference procedure for first-order logic [34].

Inference in first-order logic is only semi-decidable. Because of this, knowledge bases are often constructed using a restricted subset of first-order logic with more desirable properties. The most widely used restriction is to *Horn clauses*, which are clauses containing at most one positive literal. The Prolog programming language is based on Horn clause logic [20]. Prolog programs can be learned from databases by searching for Horn clauses that (approximately) hold in the data; this is studied in the field of inductive logic programming (ILP) [18].

Table 5.1 shows a simple KB and its conversion to clausal form. Notice that, while these formulas may be *typically* true in the real world, they are not *always* true. In most domains it is very difficult to come up with non-trivial formulas that are always true, and such formulas capture only a fraction of the relevant knowledge. Thus, despite its expressiveness, pure first-order logic has limited applicability to practical link mining problems. Many ad hoc extensions to address this have

Table 5.1 Example of a first-order knowledge base and MLN. Fr() is short for Friends(), Sm() for Smokes(), and Ca() for Cancer()

First-order logic	Clausal form	Weight
"Friends of friends are friends."		
$\forall x \forall y \forall z\ \mathtt{Fr}(x, y) \wedge \mathtt{Fr}(y, z) \Rightarrow \mathtt{Fr}(x, z)$	$\neg \mathtt{Fr}(x, y) \vee \neg \mathtt{Fr}(y, z) \vee \mathtt{Fr}(x, z)$	0.7
"Friendless people smoke."		
$\forall x\ (\neg(\exists y\ \mathtt{Fr}(x, y)) \Rightarrow \mathtt{Sm}(x))$	$\mathtt{Fr}(x, g(x)) \vee \mathtt{Sm}(x)$	2.3
"Smoking causes cancer."		
$\forall x\ \mathtt{Sm}(x) \Rightarrow \mathtt{Ca}(x)$	$\neg \mathtt{Sm}(x) \vee \mathtt{Ca}(x)$	1.5
"If two people are friends, then either		
both smoke or neither does."	$\neg \mathtt{Fr}(x, y) \vee \mathtt{Sm}(x) \vee \neg \mathtt{Sm}(y)$,	1.1
$\forall x \forall y\ \mathtt{Fr}(x, y) \Rightarrow (\mathtt{Sm}(x) \Leftrightarrow \mathtt{Sm}(y))$	$\neg \mathtt{Fr}(x, y) \vee \neg \mathtt{Sm}(x) \vee \mathtt{Sm}(y)$	1.1

[1] This conversion includes the removal of existential quantifiers by Skolemization, which is not sound in general. However, in finite domains an existentially quantified formula can simply be replaced by a disjunction of its groundings.

been proposed. In the more limited case of propositional logic, the problem is well solved by probabilistic graphical models such as Markov networks, described in the next section. We will later show how to generalize these models to the first-order case.

5.3 Markov Networks

A *Markov network* (also known as *Markov random field*) is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$ [27]. It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$P(X=x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}), \tag{5.1}$$

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). Z, known as the *partition function*, is given by $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$. Markov networks are often conveniently represented as *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x) \right). \tag{5.2}$$

A feature may be any real-valued function of the state. This chapter will focus on binary features, $f_j(x) \in \{0, 1\}$. In the most direct translation from the potential function form (5.1), there is one feature corresponding to each possible state $x_{\{k\}}$ of each clique, with its weight being $\log \phi_k(x_{\{k\}})$. This representation is exponential in the size of the cliques. However, we are free to specify a much smaller number of features (e.g., logical functions of the state of the clique), allowing for a more compact representation than the potential function form, particularly when large cliques are present. Markov logic will take advantage of this.

5.4 Markov Logic

A first-order KB can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. The basic idea in Markov logic is to soften these constraints: when a world violates one formula in the

KB it is less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight (e.g., see Table 5.1) that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.

Definition 1 [32] A Markov logic network (MLN) L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ ((5.1) and (5.2)) as follows:

1. $M_{L,C}$ contains one binary node for each possible grounding of each atom appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.
2. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

Thus there is an edge between two nodes of $M_{L,C}$ iff the corresponding ground atoms appear together in at least one grounding of one formula in L. For example, an MLN containing the formulas $\forall$x `Smokes(x)` $\Rightarrow$ `Cancer(x)` (smoking causes cancer) and $\forall$x$\forall$y `Friends(x, y)` $\Rightarrow$ (`Smokes(x)` $\Leftrightarrow$ `Smokes(y)`) (friends have similar smoking habits) applied to the constants `Anna` and `Bob` (or `A` and `B` for short) yields the ground Markov network in Fig. 5.1. Its features include `Smokes(Anna)` $\Rightarrow$ `Cancer(Anna)`, etc. Notice that, although the two formulas above are false as universally quantified logical statements, as weighted features of an MLN they capture valid statistical regularities and in fact represent a standard social network model [43]. Notice also that nodes and links in the social networks are both represented as nodes in the Markov network; arcs in the Markov network represent probabilistic dependencies between nodes and links in the social network (e.g., Anna's smoking habits depend on her friends' smoking habits).

An MLN can be viewed as a *template* for constructing Markov networks. From Definition 1 and (5.1) and (5.2), the probability distribution over possible worlds x specified by the ground Markov network $M_{L,C}$ is given by

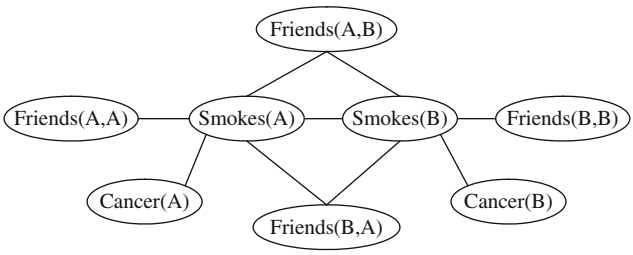


Fig. 5.1 Ground Markov network obtained by applying an MLN containing the formulas $\forall$x `Smokes(x)` $\Rightarrow$ `Cancer(x)` and $\forall$x$\forall$y `Friends(x, y)` $\Rightarrow$ (`Smokes(x)` $\Leftrightarrow$ `Smokes(y)`) to the constants `Anna(A)` and `Bob(B)`.

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right), \tag{5.3}$$

where F is the number of formulas in the MLN and $n_i(x)$ is the number of true groundings of F_i in x. As formula weights increase, an MLN increasingly resembles a purely logical KB, becoming equivalent to one in the limit of all infinite weights. When the weights are positive and finite and all formulas are simultaneously satisfiable, the satisfying solutions are the modes of the distribution represented by the ground Markov network. Most importantly, Markov logic allows contradictions between formulas, which it resolves simply by weighing the evidence on both sides.

It is interesting to see a simple example of how Markov logic generalizes first-order logic. Consider an MLN containing the single formula $\forall \mathtt{x}\ \mathtt{R(x)} \Rightarrow \mathtt{S(x)}$ with weight w and $C = \{\mathtt{A}\}$. This leads to four possible worlds: $\{\neg\mathtt{R(A)}, \neg\mathtt{S(A)}\}$, $\{\neg\mathtt{R(A)}, \mathtt{S(A)}\}$, $\{\mathtt{R(A)}, \neg\mathtt{S(A)}\}$, and $\{\mathtt{R(A)}, \mathtt{S(A)}\}$. From (5.3) we obtain that $P(\{\mathtt{R(A)}, \neg\mathtt{S(A)}\}) = 1/(3e^w + 1)$ and the probability of each of the other three worlds is $e^w/(3e^w + 1)$. (The denominator is the partition function Z; see Section 5.3.) Thus, if $w > 0$, the effect of the MLN is to make the world that is inconsistent with $\forall \mathtt{x}\ \mathtt{R(x)} \Rightarrow \mathtt{S(x)}$ less likely than the other three. From the probabilities above we obtain that $P(\mathtt{S(A)}|\mathtt{R(A)}) = 1/(1 + e^{-w})$. When $w \to \infty$, $P(\mathtt{S(A)}|\mathtt{R(A)}) \to 1$, recovering the logical entailment.

It is easily seen that all discrete probabilistic models expressible as products of potentials, including Markov networks and Bayesian networks, are expressible in Markov logic. In particular, many of the models frequently used in machine learning and data mining can be stated quite concisely as MLNs and combined and extended simply by adding the corresponding formulas. Most significantly, Markov logic facilitates the construction of non-i.i.d. models (i.e., models where objects are not independent and identically distributed). The application section shows how to describe logistic regression in Markov logic and easily extend it to perform collective classification over a set of linked objects.

When working with Markov logic, we typically make three assumptions about the logical representation: different constants refer to different objects (unique names), the only objects in the domain are those representable using the constant and function symbols (domain closure), and the value of each function for each tuple of arguments is always a known constant (known functions). These assumptions ensure that the number of possible worlds is finite and that the Markov logic network will give a well-defined probability distribution. These assumptions are quite reasonable in most practical applications and greatly simplify the use of MLNs. We will make these assumptions for the remainder of the chapter. See Richardson and Domingos [32] for further details on the Markov logic representation.

Markov logic can also be applied to a number of interesting infinite domains where some of these assumptions do not hold. See Singla and Domingos [39] for details on Markov logic in infinite domains.

For decision theoretic problems, such as the viral marketing application we will discuss later, we can easily extend MLNs to Markov logic decision networks

(MLDNs) by attaching a utility to each formula as well as a weight [25]. The utility of a world is the sum of the utilities of its satisfied formulas. Just as an MLN plus a set of constants defines a Markov network, an MLDN plus a set of constants defines a Markov decision network. The optimal decision is the setting of the action predicates that jointly maximizes expected utility.

5.5 Inference

Given an MLN model of a link mining problem, the questions of interest are answered by performing inference on it. (For example, "What are the topics of these Web pages, given the words on them and the links between them?") Recall that an MLN acts as a template for a Markov network. Therefore, we can always answer queries using standard Markov network inference methods on the instantiated network. Several of these methods have been extended to take particular advantage of the logical structure in an MLN, yielding tremendous savings in memory and time. We first provide an overview of inference in Markov networks and then describe how these methods can be adapted to take advantage of MLN structure.

5.5.1 Markov Network Inference

The main inference problem in Markov networks is computing the probabilities of query variables given some evidence and is #P-complete [35]. The most widely used method for approximate inference in Markov networks is Markov chain Monte Carlo (MCMC) [10], and in particular Gibbs sampling, which proceeds by sampling each variable in turn given its Markov blanket. (The Markov blanket of a node is the minimal set of nodes that render it independent of the remaining network; in a Markov network, this is simply the node's neighbors in the graph.) Marginal probabilities are computed by counting over these samples; conditional probabilities are computed by running the Gibbs sampler with the conditioning variables clamped to their given values.

Another popular method for inference in Markov networks is belief propagation [46]. Belief propagation is an algorithm for computing the exact marginal probability of each query variable in a tree-structured graphical model. The method consists of passing messages between variables and the potential functions they participate in. The message from a variable x to a potential function f is

$$\mu_{x \to f}(x) = \prod_{h \in nb(x) \setminus \{f\}} \mu_{h \to x}(x), \tag{5.4}$$

where $nb(x)$ is the set of potentials x appears in. The message from a potential function to a variable is

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}} \left(f(\mathbf{x}) \prod_{y\in nb(f)\setminus\{x\}} \mu_{y\to f}(y) \right), \tag{5.5}$$

where $nb(f)$ are the variables in f, and the sum is over all of these except x. In a tree, the messages from leaf variables are initialized to 1, and a pass from the leaves to the root and back to the leaves suffices. The (unnormalized) marginal of each variable x is then given by $\prod_{h\in nb(x)} \mu_{h\to x}(x)$. Evidence is incorporated by setting $f(\boldsymbol{x}) = 0$ for states $\boldsymbol{x}$ that are incompatible with it. This algorithm can still be applied when the graph has loops, repeating the message- passing until convergence. Although this *loopy* belief propagation has no guarantees of convergence or of giving accurate results, in practice it often does, and can be much more efficient than other methods.

Another basic inference task is finding the most probable state of the world given some evidence. This is known as MAP inference in the Markov network literature and MPE inference in the Bayesian network literature. (MAP means "maximum a posteriori," and MPE means "most probable explanation.") It is NP-hard. Notice that MAP inference cannot be solved simply by computing the probability of each random variable and then assigning the most probable value, because the combination of two assignments that are individually probable may itself be improbable or even impossible. Belief propagation can also be used to solve the MAP problem, by replacing summation with maximization in (5.5). Other popular methods include greedy search, simulated annealing, and graph cuts.

We first look at how to perform MAP inference and then at computing probabilities. In the remainder of this chapter, we assume that the MLN is in function-free clausal form for convenience, but these methods can be applied to other MLNs as well.

5.5.2 MAP/MPE Inference

Because of the form of (5.3) in Markov logic, the MAP inference problem reduces to finding the truth assignment that maximizes the sum of weights of satisfied clauses. This can be done using any weighted satisfiability solver and (remarkably) need not be more expensive than standard logical inference by model checking. (In fact, it can be faster, if some hard constraints are softened.) The Alchemy system uses MaxWalkSAT, a weighted variant of the WalkSAT local-search satisfiability solver, which can solve hard problems with hundreds of thousands of variables in minutes [12]. MaxWalkSAT performs this stochastic search by picking an unsatisfied clause at random and flipping the truth value of one of the atoms in it. With a certain probability, the atom is chosen randomly; otherwise, the atom is chosen to maximize the sum of satisfied clause weights when flipped. This combination of random and greedy steps allows MaxWalkSAT to avoid getting stuck in local optima while searching. Pseudocode for MaxWalkSAT is shown in Table 5.2. DeltaCost(v) computes the change in the sum of weights of unsatisfied clauses that results from

Table 5.2 MaxWalkSAT algorithm for MPE inference

```
function MaxWalkSAT(L, t_max, f_max, target, p)
    inputs: L, a set of weighted clauses
            t_max, the maximum number of tries
            f_max, the maximum number of flips
            target, target solution cost
            p, probability of taking a random step
    output: soln, best variable assignment found
vars ← variables in L
for i ← 1 to t_max
    soln ← a random truth assignment to vars
    cost ← sum of weights of unsatisfied clauses in soln
    for i ← 1 to f_max
        if cost ≤ target
            return "Success, solution is," soln
        c ← a randomly chosen unsatisfied clause
        if Uniform(0,1) < p
            v_f ← a randomly chosen variable from c
        else
            for each variable v in c
                compute DeltaCost(v)
            v_f ← v with lowest DeltaCost(v)
        soln ← soln with v_f flipped
        cost ← cost + DeltaCost(v_f)
return "Failure, best assignment is," best soln found
```

flipping variable v in the current solution. Uniform(0,1) returns a uniform deviate from the interval [0, 1].

MAP inference in Markov logic can also be performed using cutting plane methods [33] and others.

5.5.3 Marginal and Conditional Probabilities

We now consider the task of computing the probability that a formula holds, given an MLN and set of constants, and possibly other formulas as evidence. For the remainder of the chapter, we focus on the typical case where the evidence is a conjunction of ground atoms. In this scenario, further efficiency can be gained by applying a generalization of knowledge-based model construction [45]. This constructs only the minimal subset of the ground network required to answer the query and runs MCMC (or any other probabilistic inference method) on it. The network is constructed by checking if the atoms that appear in the query formula are in the evidence. If they are, the construction is complete. Those that are not are added to the network, and we in turn check the atoms they directly depend on (i.e., the atoms that appear in some formula with them). This process is repeated until all relevant atoms have been retrieved. While in the worst case it yields no savings, in practice it

can vastly reduce the time and memory required for inference. See Richardson and Domingos [32] for details.

Given the relevant ground network, inference can be performed using standard methods like MCMC and belief propagation. One problem with this is that these methods break down in the presence of deterministic or near-deterministic dependencies. Deterministic dependencies break up the space of possible worlds into regions that are not reachable from each other, violating a basic requirement of MCMC. Near-deterministic dependencies greatly slow down inference, by creating regions of low probability that are very difficult to traverse. Running multiple chains with random starting points does not solve this problem, because it does not guarantee that different regions will be sampled with frequency proportional to their probability, and there may be a very large number of regions.

We have successfully addressed this problem by combining MCMC with satisfiability testing in the MC-SAT algorithm [28]. MC-SAT is a *slice sampling* MCMC algorithm. It uses a combination of satisfiability testing and simulated annealing to sample from the slice. The advantage of using a satisfiability solver (WalkSAT) is that it efficiently finds isolated modes in the distribution, and as a result the Markov chain mixes very rapidly. The slice sampling scheme ensures that detailed balance is (approximately) preserved. MC-SAT is orders of magnitude faster than standard MCMC methods, such as Gibbs sampling and simulated tempering, and is applicable to any model that can be expressed in Markov logic.

Slice sampling [4] is an instance of a widely used approach in MCMC inference that introduces *auxiliary variables* to capture the dependencies between observed variables. For example, to sample from $P(X = x) = (1/Z) \prod_k \phi_k(x_{\{k\}})$, we can define $P(X = x, U = u) = (1/Z) \prod_k I_{[0,\phi_k(x_{\{k\}})]}(u_k)$, where ϕ_k is the kth potential function, u_k is the kth auxiliary variable, $I_{[a,b]}(u_k) = 1$ if $a \leq u_k \leq b$, and $I_{[a,b]}(u_k) = 0$ otherwise. The marginal distribution of X under this joint is $P(X = x)$, so to sample from the original distribution it suffices to sample from $P(x, u)$ and ignore the u values. $P(u_k|x)$ is uniform in $[0, \phi_k(x_{\{k\}})]$ and thus easy to sample from. The main challenge is to sample x given u, which is uniform among all $\mathcal{X}$ that satisfies $\phi_k(x_{\{k\}}) \geq u_k$ for all k. MC-SAT uses SampleSAT [44] to do this. In each sampling step, MC-SAT takes the set of all ground clauses satisfied by the current state of the world and constructs a subset, M, that must be satisfied by the next sampled state of the world. (For the moment we will assume that all clauses have positive weight.) Specifically, a satisfied ground clause is included in M with probability $1 - e^{-w}$, where w is the clause's weight. We then take as the next state a uniform sample from the set of states $SAT(M)$ that satisfy M. (Notice that $SAT(M)$ is never empty, because it always contains at least the current state.) Table 5.3 gives pseudocode for MC-SAT. $\mathcal{U}_S$ is the uniform distribution over set S. At each step, all hard clauses are selected with probability 1, and thus all sampled states satisfy them. Negative weights are handled by noting that a clause with weight $w < 0$ is equivalent to its negation with weight $-w$, and a clause's negation is the conjunction of the negations of all of its literals. Thus, instead of checking whether the clause is satisfied, we check whether its negation is satisfied; if it is, with probability $1 - e^{w}$ we select all of its negated literals, and with probability e^{w} we select none.

Table 5.3 Efficient MCMC inference algorithm for MLNs

function MC-SAT(L, n)
  inputs: L, a set of weighted clauses $\{(w_j, c_j)\}$
    n, number of samples
  output: $\{x^{(1)}, \ldots, x^{(n)}\}$, set of n samples
$x^{(0)} \leftarrow$ Satisfy(hard clauses in L)
for $i \leftarrow 1$ **to** n
  $M \leftarrow \emptyset$
  for all $(w_k, c_k) \in L$ satisfied by $x^{(i-1)}$
    With probability $1 - e^{-w_k}$ add c_k to M
  Sample $x^{(i)} \sim \mathcal{U}_{SAT(M)}$

It can be shown that MC-SAT satisfies the MCMC criteria of detailed balance and ergodicity [28], assuming a perfect uniform sampler. In general, uniform sampling is #P-hard and SampleSAT [44] only yields approximately uniform samples. However, experiments show that MC-SAT is still able to produce very accurate probability estimates, and its performance is not very sensitive to the parameter setting of SampleSAT.

5.5.4 Scaling Up Inference

5.5.4.1 Lazy Inference

One problem with the aforementioned approaches is that they require propositionalizing the domain (i.e., grounding all atoms and clauses in all possible ways), which consumes memory exponential in the arity of the clauses. Lazy inference methods [29, 38] overcome this by only grounding atoms and clauses as needed. This takes advantage of the sparseness of relational domains, where most atoms are false and most clauses are trivially satisfied. For example, in the domain of scientific research papers, most groundings of the atom `Author(person, paper)` are false and most groundings of the clause `Author(p1, paper)` $\wedge$ `Author(p2, paper)` $\Rightarrow$ `Coauthor(p1, p2)` are trivially satisfied. With lazy inference, the memory cost does not scale with the number of possible clause groundings, but only with the number of groundings that have non-default values at some point in the inference.

We first describe a general approach for making inference algorithms lazy and then show how it can be applied to create a lazy version of MaxWalkSAT. We have also developed a lazy version of MC-SAT. Working implementations of both algorithms are available in the Alchemy system. See Poon et al. [29] for more details.

Our approach depends on the concept of "default" values that occur much more frequently than others. In relational domains, the default is false for atoms and true for clauses. In a domain where most variables assume the default value, it is wasteful to allocate memory for all variables and functions in advance. The basic idea is to allocate memory only for a small subset of "active" variables and functions and

activate more if necessary as inference proceeds. In addition to saving memory, this can reduce inference time as well, since we do not allocate memory or compute values for functions that are never used.

Definition 2 Let X be the set of variables and D be their domain[2]. The default value $d^* \in D$ is the most frequent value of the variables. An evidence variable is a variable whose value is given and fixed. A function $f = f(z_1, z_2, \cdots, z_k)$ inputs z_i's, which are either variables or functions, and outputs some value in the range of f.

Although these methods can be applied to other inference algorithms, we focus on relational domains. Variables are ground atoms, which take binary values (i.e., $D = \{true, false\}$). The default value for variables is false (i.e., $d^* = false$). Examples of functions are clauses and DeltaCost in MaxWalkSAT (Table 5.2). Like variables, functions may also have default values (e.g., true for clauses). The inputs to a relational inference algorithm are a weighted KB and a set of evidence atoms (DB). Eager algorithms work by first carrying out propositionalization and then calling a propositional algorithm. In lazy inference, we directly work on the KB and DB. The following concepts are crucial to lazy inference.

Definition 3 A variable v is active iff v is set to a non-default value at some point, and x is inactive if the value of x has always been d^*. A function f is activated by a variable v if either v is an input of f or v activates a function g that is an input of f.

Let A be the eager algorithm that we want to make lazy. We make three assumptions about $\mathcal{A}$:

1. $\mathcal{A}$ updates one variable at a time. (If not, the extension is straightforward.)
2. The values of variables in $\mathcal{A}$ are properly encapsulated so that they can be accessed by the rest of the algorithm only via two methods: ReadVar(x) (which returns the value of x) and WriteVar(x, v) (which sets the value of x to v). This is reasonable given the conventions in software development, and if not, it is easy to implement.
3. $\mathcal{A}$ always sets values of variables before calling a function that depends on those variables, as it should.

To develop the lazy version of $\mathcal{A}$, we first identify the variables (usually all) and functions to make lazy. We then modify the value-accessing methods and replace the propositionalization step with lazy initialization as follows. The rest of the algorithm remains the same.

ReadVar(x): If x is in memory, Lazy-$\mathcal{A}$ returns its value as $\mathcal{A}$; otherwise, it returns d^*.

[2] For simplicity we assume that all variables have the same domain. The extension to different domains is straightforward.

WriteVar(x, v): If x is in memory, Lazy-$\mathcal{A}$ updates its value as $\mathcal{A}$. If not, and if $v = d^*$, no action is taken; otherwise, Lazy-$\mathcal{A}$ activates (allocates memory for) x and the functions activated by x, and then sets the value.

Initialization: Lazy-$\mathcal{A}$ starts by allocating memory for the lazy functions that output non-default values when all variables assume the default values. It then calls WriteVar to set values for evidence variables, which activates those evidence variables with non-default values and the functions they activate. Such variables become the initial active variables and their values are fixed throughout the inference.

Lazy-$\mathcal{A}$ carries out the same inference steps as $\mathcal{A}$ and produces the same result. It never allocates memory for more variables/functions than $\mathcal{A}$, but each access incurs slightly more overhead (in checking whether a variable or function is in memory). In the worst case, most variables are updated, and Lazy-$\mathcal{A}$ produces little savings. However, if the updates are sparse, as is the case for most algorithms in relational domains, Lazy-$\mathcal{A}$ can greatly reduce memory and time because it activates and computes the values for many fewer variables and functions.

Applying this method to MaxWalkSAT is fairly straightforward: each ground atom is a variable and each ground clause is a function to be made lazy. Following Singla and Domingos [38], we refer to the resulting algorithm as LazySAT. LazySAT initializes by activating true evidence atoms and initial unsatisfied clauses (i.e., clauses which are unsatisfied when the true evidence atoms are set to true and all other atoms are set to false)[3]. At each step in the search, the atom that is flipped is activated, as are any clauses that by definition should become active as a result. While computing DeltaCost(v), if v is active, the relevant clauses are already in memory; otherwise, they will be activated when v is set to true (a necessary step before computing the cost change when v is set to true). Table 5.4 gives pseudocode for LazySAT.

Experiments in a number of domains show that LazySAT can yield very large memory reductions, and these reductions increase with domain size [38]. For domains whose full instantiations fit in memory, running time is comparable; as problems become larger, full instantiation for MaxWalkSAT becomes impossible.

We have also used this method to implement a lazy version of MC-SAT that avoids grounding unnecessary atoms and clauses [29].

5.5.4.2 Lifted Inference

The inference methods discussed so far are purely probabilistic in the sense that they propositionalize all atoms and clauses and apply standard probabilistic inference algorithms. A key property of first-order logic is that it allows *lifted* inference, where queries are answered without materializing all the objects in the domain (e.g.,

[3] This differs from MaxWalkSAT, which assigns random values to all atoms. However, the LazySAT initialization is a valid MaxWalkSAT initialization, and the two give very similar results empirically. Given the same initialization, the two algorithms will produce exactly the same results.

Table 5.4 Lazy variant of the MaxWalkSAT algorithm

```
function LazySAT(KB, DB, t_max, f_max, target, p)
  inputs: KB, a weighted knoweldge base
          DB, database containing evidence
          t_max, the maximum number of tries
          f_max, the maximum number of flips
          target, target solution cost
          p, probability of taking a random step
  output: soln, best variable assignment found

for i ← 1 to t_max
  active_atoms ← atoms in clauses not satisfied by DB
  active_clauses ← clauses activated by active_atoms
  soln ← a random truth assignment to active_atoms
  cost ← sum of weights of unsatisfied clauses in soln
  for i ← 1 to f_max
    if cost ≤ target
      return "Success, solution is", soln
    c ← a randomly chosen unsatisfied clause
    if Uniform(0,1) < p
      v_f ← a randomly chosen variable from c
    else
      for each variable v in c
        compute DeltaCost(v), using KB if v ∉ active_atoms
      v_f ← v with lowest DeltaCost(v)
    if v_f ∉ active_atoms
      add v_f to active_atoms
      add clauses activated by v_f to active_clauses
    soln ← soln with v_f flipped
    cost ← cost + DeltaCost(v_f)
return "Failure, best assignment is", best soln found
```

resolution [34]). Lifted inference is potentially much more efficient than propositionalized inference, and extending it to probabilistic logical languages is a desirable goal. We have developed a lifted version of loopy belief propagation (BP), building on the work of Jaimovich et al. [11]. Jaimovich et al. pointed out that, if there is no evidence, BP in probabilistic logical models can be trivially lifted, because all groundings of the same atoms and clauses become indistinguishable. Our approach proceeds by identifying the subsets of atoms and clauses that remain indistinguishable even after evidence is taken into account. We then form a network with *supernodes* and *superfeatures* corresponding to these sets and apply BP to it. This network can be vastly smaller than the full ground network, with the corresponding efficiency gains. Our algorithm produces the unique minimal lifted network for every inference problem.

We begin with some necessary definitions. These assume the existence of an MLN L, set of constants C, and evidence database E (set of ground literals). For

simplicity, our definitions and explanation of the algorithm will assume that each predicate appears at most once in any given MLN clause. We will then describe how to handle multiple occurrences of a predicate in a clause.

Definition 4 *A* supernode is a set of groundings of a predicate that all send and receive the same messages at each step of belief propagation, given L, C, and E. The supernodes of a predicate form a partition of its groundings.

A *s*uperfeature is a set of groundings of a clause that all send and receive the same messages at each step of belief propagation, given L, C, and E. The superfeatures of a clause form a partition of its groundings.

Definition 5 A lifted network is a factor graph composed of supernodes and superfeatures. The factor corresponding to a superfeature $g(x)$ is $\exp(wg(x))$, where w is the weight of the corresponding first-order clause. A supernode and a superfeature have an edge between them iff some ground atom in the supernode appears in some ground clause in the superfeature. Each edge has a positive integer weight. *A* minimal lifted network is a lifted network with the smallest possible number of supernodes and superfeatures.

The first step of lifted BP is to construct the minimal lifted network. The size of this network is $O(nm)$, where n is the number of supernodes and m the number of superfeatures. In the best case, the lifted network has the same size as the MLN L and in the worst case, as the ground Markov network $M_{L,C}$.

The second and final step in lifted BP is to apply standard BP to the lifted network, with two changes:

1. The message from supernode x to superfeature f becomes

$$\mu_{f\to x}^{n(f,x)-1} \prod_{h\in nb(x)\setminus\{f\}} \mu_{h\to x}(x)^{n(h,x)},$$

   where $n(h, x)$ is the weight of the edge between h and x.
2. The (unnormalized) marginal of each supernode (and, therefore, of each ground atom in it) is given by $\prod_{h\in nb(x)} \mu_{h\to x}^{n(h,x)}(x)$.

The weight of an edge is the number of identical messages that would be sent from the ground clauses in the superfeature to each ground atom in the supernode if BP was carried out on the ground network. The $n(f, x) - 1$ exponent reflects the fact that a variable's message to a factor excludes the factor's message to the variable.

The lifted network is constructed by (essentially) simulating BP and keeping track of which ground atoms and clauses send the same messages. Initially, the groundings of each predicate fall into three groups: known true, known false, and unknown. (One or two of these may be empty.) Each such group constitutes an initial supernode. All groundings of a clause whose atoms have the same combination of truth values (true, false, or unknown) now send the same messages to the ground atoms in them. In turn, all ground atoms that receive the same number of messages from the superfeatures they appear in send the same messages and constitute a new

supernode. As the effect of the evidence propagates through the network, finer and finer supernodes and superfeatures are created.

If a clause involves predicates $R_1, \ldots, R_k$, and $N = (N_1, \ldots, N_k)$ is a corresponding tuple of supernodes, the groundings of the clause generated by N are found by joining $N_1, \ldots, N_k$ (i.e., by forming the Cartesian product of the relations $N_1, \ldots, N_k$, and selecting the tuples in which the corresponding arguments agree with each other, and with any corresponding constants in the first-order clause). Conversely, the groundings of predicate R_i connected to elements of a superfeature F are obtained by projecting F onto the arguments it shares with R_i. Lifted network construction thus proceeds by alternating between two steps:

1. Form superfeatures by doing joins of their supernodes.
2. Form supernodes by projecting superfeatures down to their predicates, and merging atoms with the same projection counts.

Pseudocode for the algorithm is shown in Table 5.5. The projection counts at convergence are the weights associated with the corresponding edges.

To handle clauses with multiple occurrences of a predicate, we keep a tuple of edge weights, one for each occurrence of the predicate in the clause. A message is passed for each occurrence of the predicate, with the corresponding edge weight. Similarly, when projecting superfeatures into supernodes, a separate count is

Table 5.5 Lifted network construction algorithm

```
function LNC(L, C, E)
   inputs: L, a Markov logic network
           C, a set of constants
           E, a set of ground literals
   output: M, a lifted network
for each predicate P
   for each truth value t in {true, false, unknown}
      form a supernode containing all groundings of P with truth value t
repeat
   for each clause involving predicates P_1, ..., P_k
      for each tuple of supernodes (N_1, ..., N_k),
         where N_i is a P_i supernode
         form a superfeature F by joining N_1, ..., N_k
   for each predicate P
      for each superfeature F it appears in
         S(P, F) ← projection of the tuples in F down to the variables in P
         for each tuple s in S(P, F)
            T(s, F) ← number of F's tuples that were projected into s
      S(P) ← ⋃_F S(P, F)
      form a new supernode from each set of tuples in S(P) with the
         same T(s, F) counts for all F
until convergence
add all current supernodes and superfeatures to M
for each supernode N and superfeature F in M
   add to M an edge between N and F with weight T(s, F)
return M
```

maintained for each occurrence, and only tuples with the same counts for all occurrences are merged.

See Singla and Domingos [40] for additional details, including the proof that this algorithm always creates the minimal lifted network.

5.6 Learning

In this section, we discuss methods for automatically learning weights, refining formulas, and constructing new formulas from data.

5.6.1 Markov Network Learning

Maximum-likelihood or MAP estimates of Markov network weights cannot be computed in closed form but, because the log-likelihood is a concave function of the weights, they can be found efficiently (modulo inference) using standard gradient-based or quasi-Newton optimization methods [26]. Another alternative is iterative scaling [6]. Features can also be learned from data, for example, by greedily constructing conjunctions of atomic features [6].

5.6.2 Generative Weight Learning

MLN weights can be learned generatively by maximizing the likelihood of a relational database (5.3). This relational database consists of one or more "possible worlds" that form our training examples. Note that we can learn to generalize from even a single example because the clause weights are shared across their many respective groundings. This is essential when the training data is a single network, such as the Web. The gradient of the log-likelihood with respect to the weights is

$$\frac{\partial}{\partial w_i} \log P_w(X = x) = n_i(x) - \sum_{x'} P_w(X = x')\, n_i(x'), \tag{5.6}$$

where the sum is over all possible databases x', and $P_w(X = x')$ is $P(X = x')$ computed using the current weight vector $w = (w_1, \ldots, w_i, \ldots)$. In other words, the ith component of the gradient is simply the difference between the number of true groundings of the ith formula in the data and its expectation according to the current model. In the generative case, even approximating these expectations tends to be prohibitively expensive or inaccurate due to the large state space. Instead, we can maximize the pseudo-likelihood of the data, a widely used alternative [1]. If x is a possible world (relational database) and x_l is the lth ground atom's truth value, the pseudo-log-likelihood of x given weights w is

$$\log P_w^*(X=x) = \sum_{l=1}^{n} \log P_w(X_l = x_l | MB_x(X_l)), \tag{5.7}$$

where $MB_x(X_l)$ is the state of X_l's Markov blanket in the data (i.e., the truth values of the ground atoms it appears in some ground formula with). Computing the pseudo-likelihood and its gradient does not require inference and is therefore much faster. Combined with the L-BFGS optimizer [19], pseudo-likelihood yields efficient learning of MLN weights even in domains with millions of ground atoms [32]. However, the pseudo-likelihood parameters may lead to poor results when long chains of inference are required.

In order to reduce overfitting, we penalize each weight with a Gaussian prior. We apply this strategy not only to generative learning but to all of our weight learning methods, even those embedded within structure learning.

5.6.3 Discriminative Weight Learning

Discriminative learning is an attractive alternative to pseudo-likelihood. In many applications, we know a priori which atoms will be evidence and which ones will be queried, and the goal is to correctly predict the latter given the former. If we partition the ground atoms in the domain into a set of evidence atoms X and a set of query atoms Y, the *conditional likelihood* of Y given X is

$$P(y|x) = \frac{1}{Z_x} \exp\left(\sum_{i \in F_Y} w_i n_i(x, y)\right) = \frac{1}{Z_x} \exp\left(\sum_{j \in G_Y} w_j g_j(x, y)\right), \tag{5.8}$$

where F_Y is the set of all MLN clauses with at least one grounding involving a query atom, $n_i(x, y)$ is the number of true groundings of the ith clause involving query atoms, G_Y is the set of ground clauses in $M_{L,C}$ involving query atoms, and $g_j(x, y) = 1$ if the jth ground clause is true in the data and 0 otherwise. The gradient of the conditional log-likelihood is

$$\begin{aligned} \frac{\partial}{\partial w_i} \log P_w(y|x) &= n_i(x, y) - \sum_{y'} P_w(y'|x) n_i(x, y') \\ &= n_i(x, y) - E_w[n_i(x, y)]. \end{aligned} \tag{5.9}$$

In the conditional case, we can approximate the expected counts $E_w[n_i(x, y)]$ using either the MAP state (i.e., the most probable state of y given x) or by averaging over several MC-SAT samples. The MAP approximation is inspired by the voted perceptron algorithm proposed by Collins [2] for discriminatively learning hidden Markov models. We can apply a similar algorithm to MLNs using MaxWalkSAT to find the approximate MAP state, following the approximate gradient for a fixed number of iterations, and averaging the weights across all iterations to combat overfitting [36].

We get the best results, however, by applying a version of the scaled conjugate gradient algorithm [24]. We use a small number of MC-SAT samples to approximate the gradient and Hessian matrix and use the inverse diagonal Hessian as a preconditioner. See Lowd and Domingos [21] for more details and results.

5.6.4 Structure Learning and Clustering

The structure of a Markov logic network is the set of formulas or clauses to which we attach weights. While these formulas are often specified by one or more experts, such knowledge is not always accurate or complete. In addition to learning weights for the provided clauses, we can revise or extend the MLN structure with new clauses learned from data. We can also learn the entire structure from scratch. The problem of discovering MLN structure is closely related to the problem of finding frequent subgraphs in graphs. Intuitively, frequent subgraphs correspond to high-probability patterns in the graph, and an MLN modeling the domain should contain formulas describing them, with the corresponding weights (unless a subgraph's probability is already well predicted by the probabilities of its subcomponents, in which case the latter suffice). More generally, MLN structure learning involves discovering patterns in hypergraphs, in the form of logical rules. The inductive logic programming (ILP) community has developed many methods for this purpose. ILP algorithms typically search for rules that have high accuracy, high coverage, etc. However, since an MLN represents a probability distribution, much better results are obtained by using an evaluation function based on pseudo-likelihood [13]. Log-likelihood or conditional log-likelihood are potentially better evaluation functions, but are much more expensive to compute. In experiments on two real-world data sets, our MLN structure learning algorithm found better MLN rules than the standard ILP algorithms CLAUDIEN [5], FOIL [30], and Aleph [41], and than a hand-written knowledge base.

MLN structure learning can start from an empty network or from an existing KB. Either way, we have found it useful to start by adding all unit clauses (single atoms) to the MLN. The weights of these capture (roughly speaking) the marginal distributions of the atoms, allowing the longer clauses to focus on modeling atom dependencies. To extend this initial model, we either repeatedly find the best clause using beam search and add it to the MLN, or add all "good" clauses of length l before trying clauses of length $l + 1$. Candidate clauses are formed by adding each predicate (negated or otherwise) to each current clause, with all possible combinations of variables, subject to the constraint that at least one variable in the new predicate must appear in the current clause. Hand-coded clauses are also modified by removing predicates.

Recently, Mihalkova and Mooney [23] introduced BUSL, an alternative, bottom-up structure learning algorithm for Markov logic. Instead of blindly constructing candidate clauses one literal at a time, they let the training data guide and constrain clause construction. First, they use a propositional Markov network structure

learner to generate a graph of relationships among atoms. Then they generate clauses from paths in this graph. In this way, BUSL focuses on clauses that have support in the training data. In experiments on three data sets, BUSL evaluated many fewer candidate clauses than our top-down algorithm, ran more quickly, and learned more accurate models.

Another key problem in MLN learning is discovering hidden variables (or inventing predicates, in the language of ILP). Link-based clustering is a special case of this, where the hidden variables are the clusters. We have developed a number of approaches for this problem and for discovering structure over the hidden variables [14–16]. The key idea is to cluster together objects that have similar relations to similar objects, cluster relations that relate similar objects, and recursively repeat this until convergence. This can be a remarkably effective approach for cleaning up and structuring a large collection of noisy linked data. For example, the SNE algorithm is able to discover thousands of clusters over millions of tuples extracted from the Web and form a semantic network from them in a few hours.

5.7 Applications

Markov logic has been applied to a wide variety of link mining problems, including link prediction (predicting academic advisors of graduate students [32]), record linkage (matching bibliographic citations [37]), link-based clustering (extracting semantic networks from the Web [15]), and many others. (See the repository of publications on the Alchemy Web site (alchemy.cs.washington.edu) for a partial list.) In this section we will discuss two illustrative examples: collective classification of Web pages and optimizing word of mouth in social networks (a.k.a. viral marketing).

5.7.1 Collective Classification

Collective classification is the task of inferring labels for a set of objects using their links as well as their attributes. For example, Web pages that link to each other tend to have similar topics. Since the labels now depend on each other, they must be inferred jointly rather than independently. In Markov logic, collective classification models can be specified with just a few formulas and applied using standard Markov logic algorithms. We demonstrate this on WebKB, one of the classic collective classification data sets [3].

WebKB consists of labeled Web pages from the computer science departments of four universities. We used the relational version of the data set from Craven and Slattery [3], which features 4165 Web pages and 10,935 Web links. Each Web page is marked with one of the following categories: student, faculty, professor, department, research project, course, or other. The goal is to predict these categories from the Web pages' words and links.

We can start with a simple logistic regression model, using only the words on the Web pages:

$$\texttt{PageClass}(\texttt{p}, +\texttt{c})$$
$$\texttt{Has}(\texttt{p}, +\texttt{w}) \Rightarrow \texttt{PageClass}(\texttt{p}, +\texttt{c})$$

The "+" notation is a shorthand for a set of rules with the same structure but different weights: the MLN contains a rule and the corresponding weight for each possible instantiation of the variables with a "+" sign. The first line, therefore, generates a unit clause for each class, capturing the prior distribution over page classes. The second line generates thousands of rules representing the relationship between each word and each class. We can encode the fact that classes are mutually exclusive and exhaustive with a set of hard (infinite-weight) constraints:

$$\texttt{PageClass}(\texttt{p}, +\texttt{c1}) \wedge (+\texttt{c1} \neq +\texttt{c2}) \Rightarrow \neg\texttt{PageClass}(\texttt{p}, +\texttt{c2})$$
$$\exists \texttt{c}\ \texttt{PageClass}(\texttt{p}, \texttt{c})$$

In Alchemy, we can instead state this property of the `PageClass` predicate in its definition using the "`!`" operator: `PageClass(page, class!)`, where page and class are type names. (In general, the "`!`" notation signifies that, for each possible combination of values of the arguments without "`!`", there is exactly one combination of the arguments with "`!`" for which the predicate is true.)

To turn this multi-class logistic regression into a collective classification model with joint inference, we only need one more formula:

$$\texttt{Linked}(\texttt{u1}, \texttt{u2}) \wedge \texttt{PageClass}(+\texttt{c1}, \texttt{u1}) \wedge \texttt{PageClass}(+\texttt{c2}, \texttt{u2})$$

This says that linked Web pages have related classes.

We performed leave-one-out cross-validation, training these models for 500 iterations of scaled conjugate gradient with a preconditioner. The logistic regression baseline had an accuracy of 70.9%, while the model with joint inference had an accuracy of 76.4%. Markov logic makes it easy to construct additional features as well, such as words on linked pages and anchor text. (See Taskar et al. [42] for a similar approach using relational Markov networks.)

5.7.2 Viral Marketing

Viral marketing is based on the premise that members of a social network influence each other's purchasing decisions. The goal is then to select the best set of people to market to, such that the overall profit is maximized by propagation of influence through the network. Originally formalized by Domingos and Richardson [8], this problem has since received much attention, including both empirical and theoretical results.

A standard data set in this area is the Epinions web of trust [31]. Epinions.com is a knowledge-sharing Web site that allows users to post and read reviews of products. The "web of trust" is formed by allowing users to maintain a list of peers whose opinions they trust. We used this network, containing 75,888 users and over 500,000 directed edges, in our experiments. With over 75,000 action nodes, this is a very large decision problem, and no general-purpose utility maximization algorithms have previously been applied to it (only domain-specific implementations).

We modeled this problem as an MLDN (Markov logic decision network) using the predicates `Buys(x)` (person x purchases the item), `Trusts(x1, x2)` (person x_1 trusts person x_2), and `MarketTo(x)` (x is targeted for marketing). `MarketTo(x)` is an *action predicate*, since it can be manipulated directly, whereas `Buys(x)` and `Trusts(x1, x2)` are *state predicates*, since they cannot. The utility function is represented by the unit clauses `Buys(x)` (with positive utility, representing profits from sales) and `MarketTo(x)` (with negative utility, representing the cost of marketing). The topology of the social network is specified by an evidence database of `Trusts(x1, x2)` atoms.

The core of the model consists of two formulas:

$$\texttt{Buys}(x_1) \wedge \texttt{Trusts}(x_2, x_1) \Rightarrow \texttt{Buys}(x_2)$$
$$\texttt{MarketTo}(+x) \Rightarrow \texttt{Buys}(x)$$

The weight of the first formula represents how strongly x_1 influences x_2, and the weight of the second formula represents how strongly users are influenced by marketing. In addition, the model includes the unit clause `Buys(x)` with a negative weight, representing the fact that most users do not buy most products. The final model is very similar to that of Domingos and Richardson [8] and yields comparable results, but Markov logic makes it much easier to specify. Unlike previous hand-coded models, our MLDN can be easily extended to incorporate customer and product attributes, purchase history information, multiple types of relationships, products, actors in the network, marketing actions, etc. Doing so is a direction for future work. See Nath and Domingos [25] for additional details.

5.8 The Alchemy System

The inference and learning algorithms described in the previous sections are publicly available in the open-source Alchemy system [17]. Alchemy makes it possible to define sophisticated probabilistic models over relational domains with a few formulas, learn them from data, and use them for prediction, understanding, etc. From the user's point of view, Alchemy makes it easier and quicker to develop link-mining applications by taking advantage of the Markov logic language and the existing library of algorithms for it. From the researcher's point of view, Alchemy makes it possible to easily integrate new algorithms with a full complement of other algorithms that support them or make use of them, and to make the new algorithms

available for a wide variety of applications without having to target each one individually.

Alchemy can be viewed as a declarative programming language akin to Prolog, but with a number of key differences: the underlying inference mechanism is model checking instead of theorem proving; the full syntax of first-order logic is allowed, rather than just Horn clauses; and, most importantly, the ability to handle uncertainty and learn from data is already built in. Table 5.6 compares Alchemy with Prolog and BUGS [22], one of the most popular toolkits for Bayesian modeling and inference.

Table 5.6 A comparison of Alchemy, Prolog, and BUGS

Aspect	Alchemy	Prolog	BUGS
Representation	First-order logic + Markov nets	Horn clauses	Bayes nets
Inference	SAT, MCMC, lifted BP	Theorem proving	MCMC
Learning	Parameters and structure	No	Parameters
Uncertainty	Yes	No	Yes
Relational	Yes	Yes	No

5.9 Conclusion and Directions for Future Research

Markov logic offers a simple yet powerful representation for link mining problems. Since it generalizes first-order logic, Markov logic can easily model the full relational structure of link mining problems, including multiple relations and attributes of different types and arities, relational concepts such as transitivity, and background knowledge in first-order logic. And since it generalizes probabilistic graphical models, Markov logic can efficiently represent uncertainty in the attributes, links, cluster memberships, etc., required by most link mining applications.

The specification of standard link mining problems in Markov logic is remarkably compact, and the open-source Alchemy system (available at alchemy.cs.washington.edu) provides a powerful set of algorithms for solving them. We hope that Markov logic and Alchemy will be of use to link mining researchers and practitioners who wish to have the full spectrum of logical and statistical inference and learning techniques at their disposal, without having to develop every piece themselves. More details on Markov logic and its applications can be found in Domingos and Lowd [7].

Directions for future research in Markov logic include further increasing the scalability, robustness and ease of use of the algorithms, applying it to new link mining problems, developing new capabilities, etc.

Acknowledgments This research was partly supported by ARO grant W911NF-08-1-0242, DARPA contracts FA8750-05-2-0283, FA8750-07-D-0185, HR0011-06-C-0025, HR0011-07-C-0060 and NBCH-D030010, NSF grants IIS-0534881 and IIS-0803481, ONR grants N-00014-05-1-0313 and N00014-08-1-0670, an NSF CAREER Award (first author), a Sloan Research Fellowship (first author), an NSF Graduate Fellowship (second author), and a Microsoft Research Graduate Fellowship (second author). The views and conclusions contained in this document are those of

the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, DARPA, NSF, ONR, or the US Government.

References

1. J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24:179–195, 1975.
2. M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, ACL Philadelphia, PA, 2002.
3. M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1/2):97–119, 2001.
4. P. Damien, J. Wakefield, and S. Walker. Gibbs sampling for Bayesian non-conjugate and hierarchical models by auxiliary variables. *Journal of the Royal Statistical Society, Series B*, 61:331–344, 1999.
5. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
6. S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–392, 1997.
7. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI*. Morgan & Claypool, San Rafael, CA, 2009.
8. P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 57–66, ACM Press, San Francisco, CA, 2001.
9. M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.
10. W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London, 1996.
11. A. Jaimovich, O. Meshi, and N. Friedman. Template based inference in symmetric relational Markov random fields. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 191–199, AUAI Press, Vancouver, BC, 2007.
12. H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications*, pages 573–586. American Mathematical Society, New York, NY, 1997.
13. S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 441–448, ACM Press, Bonn, 2005.
14. S. Kok and P. Domingos. Statistical predicate invention. In *Proceedings of the 24th International Conference on Machine Learning*, pages 433–440, ACM Press, Corvallis, OR, 2007.
15. S. Kok and P. Domingos. Extracting semantic networks from text via relational clustering. In *Proceedings of the 19th European Conference on Machine Learning*, pages 624–639, Springer, Antwerp, 2008.
16. S. Kok and P. Domingos. Hypergraph lifting for structure learning in Markov logic networks. In *Proceedings of the 26th International Conference on Machine Learning*, ACM Press, Montreal, QC, 2009.
17. S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2007. http://alchemy.cs.washington.edu.
18. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, 1994.

19. D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.
20. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1987.
21. D. Lowd and P. Domingos. Efficient weight learning for Markov logic networks. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 200–211, Springer, Warsaw, 2007.
22. D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter. WinBUGS — a Bayesian modeling framework: concepts, structure, and extensibility. *Statistics and Computing*, 10:325–337, 2000.
23. L. Mihalkova and R. Mooney. Bottom-up learning of Markov logic network structure. In *Proceedings of the 24th International Conference on Machine Learning*, pages 625–632, ACM Press, Corvallis, OR, 2007.
24. M. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.
25. A. Nath and P. Domingos. A language for relational decision theory. In *Proceedings of the International Workshop on Statistical Relational Learning*, Leuven, 2009.
26. J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, NY, 2006.
27. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, 1988.
28. H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 458–463, AAAI Press, Boston, MA, 2006.
29. H. Poon, P. Domingos, and M. Sumner. A general method for reducing the complexity of relational inference and its application to MCMC. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, pages 1075–1080, AAAI Press, Chicago, IL, 2008.
30. J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
31. M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 61–70, ACM Press, Edmonton, AB, 2002.
32. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.
33. S. Riedel. Improving the accuracy and efficiency of MAP inference for Markov logic. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 468–475, AUAI Press, Helsinki, 2008.
34. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.
35. D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.
36. P. Singla and P. Domingos. Discriminative training of Markov logic networks. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 868–873, AAAI Press, Pittsburgh, PA, 2005.
37. P. Singla and P. Domingos. Entity resolution with Markov logic. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pages 572–582, IEEE Computer Society Press, Hong Kong, 2006.
38. P. Singla and P. Domingos. Memory-efficient inference in relational domains. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 488–493, AAAI Press, Boston, MA, 2006.
39. P. Singla and P. Domingos. Markov logic in infinite domains. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 368–375, AUAI Press, Vancouver, BC, 2007.
40. P. Singla and P. Domingos. Lifted first-order belief propagation. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, pages 1094–1099, AAAI Press, Chicago, IL, 2008.

41. A. Srinivasan. The Aleph manual. Technical report, Computing Laboratory, Oxford University, 2000.
42. B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pages 485–492, Morgan Kaufmann, Edmonton, AB, 2002.
43. S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, 1994.
44. W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: Exploiting random walk strategies. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 670–676, AAAI Press, San Jose, CA, 2004.
45. M. Wellman, J. S. Breese, and R. P. Goldman. From knowledge bases to decision models. *Knowledge Engineering Review*, 7:35–53, 1992.
46. J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 689–695. MIT Press, Cambridge, MA, 2001.

Chapter 6
Understanding Group Structures and Properties in Social Media

Lei Tang and Huan Liu

Abstract The rapid growth of social networking sites enables people to connect to each other more conveniently than ever. With easy-to-use social media, people contribute and consume contents, leading to a new form of human interaction and the emergence of online collective behavior. In this chapter, we aim to understand group structures and properties by extracting and profiling communities in social media. We present some challenges of community detection in social media. A prominent one is that networks in social media are often heterogeneous. We introduce two types of heterogeneity presented in online social networks and elaborate corresponding community detection approaches for each type, respectively. Social media provides not only interaction information but also textual and tag data. This variety of data can be exploited to profile individual groups in understanding group formation and relationships. We also suggest some future work in understanding group structures and properties.

6.1 Introduction

Social media such as Facebook, MySpace, Twitter, and BlogSpot facilitates people of all walks of life to express their thoughts, voice their opinions, and connect to each other anytime and anywhere. For instance, popular content-sharing sites like Del.icio.us, Flickr, and YouTube allow users to upload, tag, and comment different types of contents (bookmarks, photos, videos). Users registered at these sites can also become friends, a fan, or a follower of others. Social media offers rich information of human interaction and collective behavior in a much larger scale (hundreds of thousands or millions of actors). It is gaining increasing attention across various disciplines including sociology, behavior science, anthropology, computer science, epidemics, economics, marketing business, to name a few.

H. Liu (✉)
Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-8809, USA
e-mail: huanliu@asu.edu

L. Tang
Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-8809, USA
e-mail: l.tang@asu.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_6,

With the expanded use of Web and social media, virtual communities and online interactions have become a vital part of human experience. Members of virtual communities tend to share similar interests or topics and connect to each other in a community more frequently than with those outside the community. For example, there can be two groups browsing news at a Web site, say *digg.com*: one is interested in topics related to *Meteorology*, while the other in *Politics*; A blogger (say the owner of *http://hunch.net/*) who publishes blog posts actively on "machine learning" often has links on his/her blog site to other bloggers who concentrate on "machine learning" as well. It would be interesting to find these like-minded individuals for developing many other applications to enhance personal experience or to improve business intelligence. In this work, we focus on *communities* (or equivalently *groups*) in social media. There is a wide range of applications of discovering groups (a.k.a. *community detection*) based on the interactions among actors and capturing group properties via shared topics, including visualization [8], recommendation and classification [18, 19], influence study [1], direct marketing, group tracking, and recommendation.

Community detection is a classical task in social network analysis. However, some new features presented in networks of social media entail novel solutions to handle online communities.

- *Heterogeneity*. Networks in social media tend to involve multiple types of entities or interactions. For instance, in content-sharing sites like Flickr and YouTube, multiple types of entities: users, tags, comments, and contents are intertwined with each other. Sometimes, users at the same social network site can interact with each other in various forms, leading to heterogeneous types of interactions between them. It is intriguing to explore whether or not heterogeneous information can help identify communities. It is also challenging to effectively fuse these heterogeneous types of information.
- *Large-Scale Networks*. Networks in social media are typically in a much larger scale than those in traditional social network analysis. Traditional social network analysis relies on circulation of questionnaires or surveys to collect interaction information of human subjects, limiting the scale of analysis to hundreds of actors mostly. Hence, scalability is seldom a focus there. Networks in social media, on the contrary, involve a much larger number of actors, which presents a challenge of scalability. In addition, large-scale networks yield similar patterns, such as power-law distribution for node degrees and small-world effect [3]. It is yet unclear how these patterns can help or guide data mining tasks.
- *Collective Intelligence*. In social media, crowd wisdom, in forms of tags and comments, is often available. Is it possible to employ collective intelligence to help understand group structures and properties? For instance, how to characterize a group? How to differentiate a group from others in social media? What are potential causes that lead some users to form a community? With abounding groups in social media, how can we understand the relationship among them?
- *Evolution*. Each day in social media, new users join the network and new connections occur between existing members, while some existing ones leave or become

dormant. How can we capture the dynamics of individuals in networks? Can we find the members that act like the backbone of communities? The group interests might change as well. How can we update the group interests and relations accordingly as information evolves?

Given the features above, we will mainly discuss two research issues concerning communities in social media: (1) identifying communities in social media via the readily- available interaction information; and (2) profiling groups dynamically using descriptive tags and taxonomy adaptation. The two research tasks are highly related to each other. The first task identifies groups, serving as the basis for the second one; and the second task helps understand the formation of identified groups and unravel properties why users join together to form a group. In the following section, we first introduce heterogeneous networks in social media and define the problems of interest and motivations. We will then elucidate the technical details with challenges and solutions for both tasks in the subsequent sections.

6.2 Heterogeneous Networks in Social Media

There are two types of heterogeneous networks that demand special attention. We first illustrate the two types and then expound the necessity for considering heterogeneity in community detection.

6.2.1 Heterogeneous Networks

With social media, people can connect to each other more conveniently than ever. In some social networking sites, entities other than human beings can also be involved. For instance, in YouTube, a user can upload a video and another user can tag it. In other words, the users, videos, and tags are weaved into the same network. The "actors" in the network are not at all homogeneous. Furthermore, examining activities of users, we can observe different interaction networks between the same set of actors. Take YouTube again as an example. A user can become a friend of another user's; he can also subscribe to another user. The existence of different relations suggests that the interactions between actors are heterogeneous. Networks involving heterogeneous actors or interactions are referred as *heterogeneous networks*. Accordingly, heterogeneous networks can be categorized in two different types:

– *Multi-mode Networks* [22]. A multi-mode network involves heterogeneous actors. Each mode represents one type of entity. For instance, in the YouTube example above, a three-mode network can be constructed, with videos, tags, and users each representing a mode, as seen in Fig. 6.1. There are disparate interactions among the three types of entities: users can upload videos. They can also provide tags for some videos. Intuitively, two users contributing similar videos

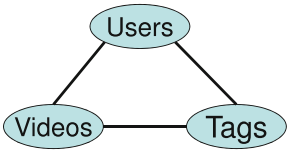


Fig. 6.1 A multi-mode network in YouTube

or tags are likely to share interests. Videos sharing similar tags or users are more likely to be related. Note that in the network, both tags and videos are also considered as "actors," though users are probably the major mode under consideration.

Other domains involving networks or interactions also encounter multi-mode networks. An example of multi-mode network is academic publications as shown in Fig. 6.2. Various kinds of entities (researchers, conferences/journals, papers, words) are considered. Scientific literature connects papers by citations; papers are published at different places (conferences, journals, workshops, thesis, etc.); and researchers are connected to papers through authorship. Some might relate to each other by serving simultaneously as journal editors or on conference program committees. Moreover, each paper can focus on different topics, which are represented by words. Words are associated to each other based on semantics. At the same time, papers connect to different conferences, journals (venues for publication). In the network, there are multiple types of entities. And entities relate to others (either the same type or different types) through different links.

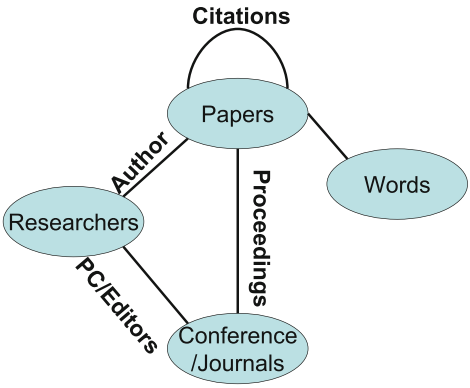


Fig. 6.2 A multi-mode network in academia

– *Multi-dimensional Networks* [20, 23]. A multi-dimensional network has multiple types of interactions between the same set of users. Each dimension of the network represents one type of activity between users. For instance, in Fig. 6.3, at popular photo and video sharing sites (e.g., Flickr and YouTube), a user can connect to his friends through email invitation or the provided "add as contacts" function; users can also tag/comment on the social contents like photos and

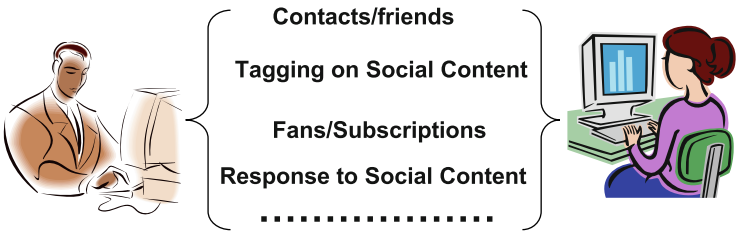


Fig. 6.3 An example of multi-dimensional network

videos; a user at YouTube can respond to another user by uploading a video; and a user can also become a fan of another user by subscription to the user's contributions of social contents. A network among these users can be constructed based on each form of activity, in which *each dimension represents one facet of diverse interaction*.

Actually, directed networks can be considered as a special case of multi-dimensional network. Take email communications as an example. People can play two different roles in email communications: senders and receivers. These two roles are not interchangeable. Spammers send an overwhelming number of emails to normal users but seldom receive responses from them. The sender and receiver roles essentially represent two different interaction patterns. A two-dimensional network can be constructed to capture the roles of senders and receivers. In the first dimension, two actors are deemed related if they both send emails to the same person; in the other dimension, two actors interact if they both receive emails from another actor. A similar idea is also adopted as "hubs" and "authorities" on Web pages [10].

In this chapter, we do not use the notion of *multi-relational network*, as "multi-relational" has been used with different connotations depending on the domains. For example, multi-relational data mining [4], originating from the database field, focuses on data mining tasks with multiple relational tables. This concept can be extended to networks as well. One special case is that, each table is considered as interactions of two types of entities, leading to a multi-mode network. Meanwhile, social scientists [27] use multi-relational network for a different meaning. A multi-relational network is a network in which the connections (edges) between actors represent different type of relations, e.g., father-of, wife-of. If each type of interaction in a multi-dimensional network represents one relation, the multi-dimensional network is equivalent to a multi-relational network.

Note that the two types of heterogeneous networks (multi-mode and multi-dimensional) mentioned above are not exclusive. A complicated network can be both multi-mode and multi-dimensional at the same time. As presented later, techniques to address these two types of networks can be fused together for community discovery.

6.2.2 Motivations to Study Network Heterogeneity

Social media offers an easily accessible platform for diverse online social activities and also introduces heterogeneity in networks. Thus, it calls for solutions to extract communities in heterogeneous networks, which will be covered in the next section. However, it remains unanswered why one cannot reduce a heterogeneous network to several homogeneous ones (i.e., one mode or one dimension) for investigation.

The reason is that the interaction information in one mode or one dimension might be too noisy to detect meaningful communities. For instance, in the YouTube example in Fig. 6.1, it seems acceptable if we only consider the user mode. In other words, just study the friendship network. On the one hand, some users might not have any online friends either because they are too introvert to talk to other online users, or because they just join the network and are not ready for or not interested in connections. On the other hand, some users might abuse connections, since it is relatively easy to make connections in social media compared with the physical world. As mentioned in [18], a user in Flickr can have thousands of friends. This can hardly be true in the real world. It might be the case that two online users get connected but they never talk to each other. Thus, these online connections of one mode or one dimension can hardly paint a true picture of what is happening.

A single type of interaction provides limited (often sparse) information about the community membership of online users. Fortunately, social media provides more than just a single friendship network. A user might engage in other forms of activities besides connecting to friends. It is helpful to utilize information from other modes or dimensions for more effective community detection. It is empirically verified that communities extracted using multi-mode or multi-dimensional information are more accurate and robust [23].

6.3 Community Extraction in Heterogeneous Networks

We first formulate the community detection problems for multi-mode networks and multi-dimensional networks, respectively; and then present viable solutions and their connections.

6.3.1 Multi-mode Networks

Given an m-mode network with m types of actors

$$\mathbb{X}_i = \left\{x_1^i, x_2^i, \cdots, x_{n_i}^i\right\} \quad i = 1, \cdots, m$$

where n_i is the number of actors for $\mathbb{X}_i$, we aim to find community structures in each mode. Let $R_{i,j} \in \mathbb{R}^{n_i \times n_j}$ denote the interaction between two modes of actors $\mathbb{X}_i$ and $\mathbb{X}_j$, k_i and k_j denote the number of latent communities for $\mathbb{X}_i$ and $\mathbb{X}_j$, respectively

Table 6.1 Notations

Symbol	Representation
m	number of modes in a multi-mode network
n_i	number of actors in mode i
k_i	number of communities at mode i
$R_{i,j}$	interaction matrix between modes i and j
C_i	community indicator matrix of mode i
$A_{i,j}$	group interaction density between modes i and j
c_{st}^i	the (s,t)th entry of C_i
$\mathcal{R}$	a multi-dimensional network
R_d	the dth dimension of multi-dimensional network
n	number of actors within a multi-dimensional network
d	the dimensionality of a multi-dimensional network
k	number of communities within a network
C	the community indicator matrix

(Table 6.1). The interactions between actors can be approximated by the interactions between groups in the following form [12]:

$$R_{i,j} \approx C_i \; A_{i,j} \; C_j^T,$$

where $C_i \in \{0,1\}^{n_i \times k_i}$ denotes some latent cluster membership for $\mathbb{X}_i$, $A_{i,j}$ the group interaction, and T the transpose of a matrix. In other words, the group identity determines how two actors interact, essentially making a similar assumption as that of block models [17]. The difference is that block models deal with the problem from a probabilistic aspect and concentrate on one-mode or two-mode networks. Here we try to identify the block structure of multi-mode networks via matrix approximation:

$$\min \sum_{1 \le i < j \le m} w_{ij} \| R_{i,j} - C_i A_{i,j} C_j^T \|_F^2, \tag{6.1}$$

$$s.t. \; C_i \in \{0,1\}^{n_i \times k_i} \quad i = 1, 2, \ldots, m, \tag{6.2}$$

$$\sum_{t=1}^{k_i} c_{st}^i = 1, \quad s = 1, 2, \ldots, n_i, \quad i = 1, 2, \ldots, m, \tag{6.3}$$

where w_{ij} are the weights associated with different interactions and c_{st}^i the (s,t)th entry of C_i.

The constraints in (6.3) force each row of the indicator matrix to have only one entry being 1. That is, each actor belongs to only one community. Unfortunately, the discreteness of the constraints in (6.2) makes the problem NP-hard. A strategy that has been well studied in spectral clustering is to allow the cluster indicator matrix to be continuous and relax the hard clustering constraint as follows:

$$C_i^T C_i = I_{k_i}, \quad i = 1, 2, \ldots, m. \tag{6.4}$$

This continuous approximation of C_i can be considered as a low-dimensional embedding such that the community structure is more prominent in these dimensions. Consequently, the problem can be reformulated as

$$\min_{C,\ A} \sum_{1 \le i < j \le m} w_{ij} \| R_{i,j} - C_i\ A_{i,j}\ C_j^T \|_F^2 \tag{6.5}$$

$$s.t. \quad C_i^T C_i = I_{k_i}, \quad i = 1, 2, \ldots, m. \tag{6.6}$$

Since the solution of C_i of the above formulation is continuous, a post-processing step is required to obtain the disjoint partition of actors. A commonly used technique is to treat each column of C_i as features and then conduct k-means clustering to obtain discrete assignment of clusters [13]. Below, we briefly describe the computation of $A_{i,j}$ and C_i in (6.5).

Note that the problem in (6.5) is too complicated to derive a closed-form solution. However, it can be solved iteratively. First, we show that $A_{i,j}$ has a closed-form solution when C_i is fixed. Then, we plug in the optimal $A_{i,j}$ and compute C_i via alternating optimization. Basically, fix the community indicator at all other modes while computing the community indicator C_i at mode i. We only include the key proof here due to the space limit. Please refer to [12, 22] for details.

Theorem 1 *Given C_i and C_j, the optimal group interaction matrix $A_{i,j}$ can be calculated as*

$$A_{i,j} = C_i^T R_{i,j} C_j. \tag{6.7}$$

Proof Since $A_{i,j}$ appears only in a single term, we can focus on the term to optimize $A_{i,j}$.

$$\begin{aligned} & \| R_{i,j} - C_i A_{i,j} C_j^T \|_F^2 \\ = & \, tr \left[\left(R_{i,j} - C_i A_{i,j} C_j^T \right) \left(R_{i,j} - C_i A_{i,j} C_j^T \right)^T \right] \\ = & \, tr \left[R_{i,j} R_{i,j}^T - 2 C_i A_{i,j} C_j^T R_{i,j}^T + A_{i,j} A_{i,j}^T \right] \end{aligned}$$

The second equation is obtained based on the property that $tr(AB) = tr(BA)$ and column orthogonality of C_i and C_j. Setting the derivative with respect to $A_{i,j}$ to zero, we have $A_{i,j} = C_i^T R_{i,j} C_j$. The proof is completed. □

Given the optimal $A_{i,j}$ as in (6.7), it can be verified that

$$\| R_{i,j} - C_i A_{i,j} C_j^T \|_F^2 = \| R_{i,j} \|_F^2 - \| C_i^T R_{i,j} C_j \|_F^2. \tag{6.8}$$

Since $\| R_{i,j} \|_F^2$ in (6.8) are constants, we can transform the formulation in Eq. (6.5) into the following objective:

$$\max \sum_{1 \le i < j \le m}^{m} w_{i,j} \| C_i^T R_{i,j} C_j \|_F^2 \tag{6.9}$$

$$s.t.\ C_i^T C_i = I_{k_i}, \quad i = 1, 2, \ldots, m \tag{6.10}$$

Note that C_i is interrelated with C_j $(j \neq i)$. There is no closed-form solution in general. However, given C_j $(j \neq i)$, the optimal C_i can be computed as follows:

Theorem 2 *Given C_j $(j \neq i)$, C_i can be computed as the top left singular vectors of the matrix P_i concatenated by the following matrices in column-wise:*

$$P_i = \left[\left\{ \sqrt{w_{ij}}\, R_{i,j} C_j \right\}_{i<j}, \quad \left\{ \sqrt{w_{ki}}\, R_{k,i}^T C_k \right\}_{k<i} \right]. \tag{6.11}$$

Proof We only focus on those terms in the objective involving C_i.

$$\begin{aligned} L &= \sum_{i<j} w_{ij} \| C_i^T R_{i,j} C_j \|_F^2 + \sum_{k<i} w_{ki} \| C_k^T R_{k,i} C_i \|_F^2 \\ &= \sum_{i<j} w_{ij}\, tr\left(C_i^T R_{i,j} C_j C_j^T R_{i,j}^T C_i \right) + \sum_{k<i} w_{ki}\, tr\left(C_i^T R_{k,i}^T C_k C_k^T R_{k,i} C_i \right) \\ &= tr\left[C_i^T \left(\sum_{i<j} w_{ij} R_{i,j} C_j C_j^T R_{i,j}^T + \sum_{k<i} w_{ki} R_{k,i}^T C_k C_k^T R_{k,i} \right) C_i \right] \\ &= tr\left(C_i^T M_i C_i \right), \end{aligned}$$

where M_i is defined as

$$M_i = \sum_{i<j} w_{ij} R_{i,j} C_j C_j^T R_{i,j}^T + \sum_{k<i} w_{ki} R_{k,i}^T C_k C_k^T R_{k,i}. \tag{6.12}$$

So the problem boils down to a well-defined max-trace problem with orthogonality constraints. The community indicator matrix C_i has a closed-form solution, which corresponds to the subspace spanned by the top k_i eigenvectors of M_i. Note that M_i is normally a dense $n_i \times n_i$ matrix. Direct calculation of M_i and its eigenvectors is expensive if n_i is huge (which is typically true in social media). However, M_i can be written as

$$M_i = P_i P_i^T, \tag{6.13}$$

where P_i is defined as in (6.11). Thus the optimal C_i, which corresponds to the top eigenvectors of M_i can be computed as the top left singular vectors of P_i. Note that the ordering of columns in P_i does not affect the final solution. □

As can be seen in (6.11), the clustering results of interacted entities essentially form weighted features for clustering of the ith mode. The matrix M_i, being the

outer product of P_i, acts like a similarity matrix for clustering. Based on Theorem 2, we can update the cluster indicator matrix iteratively based on the "attributes" obtained from the clustering results of related entities.

Once the approximate cluster indicator matrix C_i is computed, k-means can be applied to obtain the discrete assignment of communities for actors at each mode. The overall description of the algorithm is presented in Fig. 6.4. In the algorithm, we specify the objective to be calculated via (6.9), as the direct calculation of the original formation in (6.5) usually requires computation of dense matrices, which is not applicable for large-scale multi-mode networks.

Input: $\{R_{i,j}\}$, $\{k_i\}$, $\{w_{ij}\}$
Output: $\{idx_i\}$, $\{C_i\}$, $\{A_{i,j}\}$
1. Generate initial cluster indicator matrix $\{C_i\}$.
2. **Repeat**
3. **For** $i = 1, 2, \ldots, m$
4. construct P_i as in Eq. (11);
5. update C_i as top k_i left singular vectors of P_i;
6. **Until** the relative change of the objective in Eq. (9) $\leq \epsilon$.
7. calculate $A_{i,j}$ as in Eq. (7)
8. calculate the cluster idx_i with k-means on C_i

Fig. 6.4 Algorithm for community extraction in multi-mode networks

6.3.2 Multi-dimensional Networks

In a multi-dimensional network, there are multiple dimensions of interactions between the same set of users. A latent community structure in social media exists among these actors, indicating various interactions along different dimensions. The goal of community extraction in a multi-dimensional network is to infer the shared community structure. A d-dimensional network is represented as

$$\mathcal{R} = \{R_1, R_2, \ldots, R_d\}.$$

R_i represents the interactions among actors in the ith dimension. For simplicity, we assume the interaction matrix R_i is symmetric. We use $C \in \{0, 1\}^{n \times k}$ to denote the community membership of each actor.

Since the goal of community extraction in multi-dimensional networks is to identify a shared community structure that explains the interaction in each dimension, one straightforward approach is to average the interaction in each dimension, and treat it as a normal single-dimensional network. Then, any community extraction methods proposed for networks or graphs can be applied. This simple averaging approach becomes problematic if the interaction in each dimension is not directly comparable. For example, it can be the case that users interact with each other frequently in one dimension (say, leave some comments on friend's photos), whereas

talk to each other less frequently in another dimension (say, sending emails in Facebook). Averaging the two types of interaction might misrepresent the hidden community information beneath the latter dimension with less frequent interactions. One way to alleviate this problem is to assign different weights for each dimension. Unfortunately, it is not an easy task to assign appropriate weights for effective community extraction.

Another variant is to optimize certain averaged clustering criteria. Let $Q_i(C)$ denote the cost of community structure C on the ith dimension of interaction R_i. We list some representative criteria in existing body of literature as follows:

- Block model approximation [28] minimizes the divergence of the interaction matrix and block model approximation:

$$\min Q = \ell\left(R; C^T \Lambda C\right) \tag{6.14}$$

  where ℓ is a loss function to measure the difference of two matrices, and Λ a diagonal matrix roughly representing the within-group interaction density.
- Spectral clustering [13] minimizes the following cost function

$$\min Q = tr\left(C^T L C\right), \tag{6.15}$$

  where L is the graph Laplacian.
- Modularity maximization [15] maximizes the modularity of a community assignment:

$$\max Q = tr\left(C^T B C\right), \tag{6.16}$$

  where B is a modularity matrix.

Given a multi-dimensional network, we can optimize the following cost function,

$$\min_C \sum_{i=1}^{d} w_i Q_i(C). \tag{6.17}$$

The weighted optimization criterion with graph Laplacian and random walk interpretation are presented in [29]. Weighted modularity maximization is explored in [23] as a baseline approach.

The drawback of the aforementioned two approaches (averaging network interactions or minimize average cost) is that they can be sensitive to noisy interactions. Assigning proper weights can help alleviate the problem, but it is equally, if not more, difficult to choose a good heuristic of weighting scheme. Instead, an alternative paradigm based on structural features is proposed in [23] to overcome these disadvantages. The basic idea is that the community structure extracted from each dimension of the network should be similar. Hence, we can extract the "rough"

community structure at each dimension, and then integrate them all to find out the shared community structure. Thus, the paradigm consists of two phases: (i) *structural feature extraction from each dimension* and (ii) *cross-dimension integration*.

– *Phase I: Structural Feature Extraction* Structural features, which are indicative of some community structure, are extracted based on network connectivity. Any methods finding out a community assignment can be used to extract structural features. Note that finding out a discrete assignment of clusters with respect to the criteria in (6.14), (6.15), and (6.16) is NP-complete. Commonly used algorithms are very sensitive to network topology [7] and suffer from local optima. In practice, some approximation scheme of the discrete assignment is often exploited.

  One widely used relaxation, as we have done in the previous section, is to allow C to be continuous while satisfying certain orthogonal constraints (i.e., $C^T C = I_k$). This relaxation results in an approximation of C which can be considered as a lower dimensional embedding that captures the community structure. The optimal C typically corresponds to the top eigenvectors of a certain matrix. This relaxation is adopted in [15] for modularity maximization and many spectral clustering approaches [13]. Note that after relaxation, the obtained community indicator matrix C is typically globally optimal with respect to certain criteria. This avoids the randomness of a discrete assignment due to the noise in network connections or algorithm initialization. Hence, structural feature extraction based on relaxed community indicator is a more favorable solution. Networks in social media are very noisy. Extracting some prominent structural features indeed helps remove the noise, enabling more accurate community identification in the second stage.
– *Phase II: Cross-Dimension Integration* Assuming a latent community structure is shared across dimensions in a multi-dimensional network, we expect that the extracted structural features to be "similar." However, dissimilar structural feature values do not necessarily indicate that the corresponding community structures are different as an orthogonal transformation or reordering of columns in C can be "equivalent" solutions [23]. Instead, we expect the structural features of different dimensions to be highly correlated after certain transformation. Thus, the integration boils down to finding transformations that can be applied to the extracted structural features to maximize the correlation.

  To capture the correlations between multiple sets of variables (generalized), canonical correlation analysis (CCA) [9] is a standard statistical technique. CCA attempts to find a linear transformation for each set of variables such that the pairwise correlations are maximized. It has been widely used to integrate information from multiple different sources or views [6, 16]. Here we briefly illustrate one scheme of generalized CCA that turns out to equal to principal component analysis (PCA) under certain constraints.

  Let $C_i \in R^{n \times \ell_i}$ denote the ℓ_i structural features extracted from the ith dimension of the network, and $w_i \in \mathbb{R}^{\ell_i}$ be the linear transformation applied to the structural features of network dimension i. The correlation between two dimensions after transformation is

$$(C_i w_i)^T (C_j w_j) = w_i^T \left(C_i^T C_j\right) w_j = w_i^T O_{ij} w_j,$$

with $O_{ij} = C_i^T C_j$ representing the covariance between the structural features of the ith and the jth dimensions. One scheme of generalized CCA attempts to maximize the summation of pairwise correlations in the following form:

$$\max \sum_{i=1}^{d} \sum_{j=1}^{d} w_i^T O_{ij} w_j \tag{6.18}$$

$$s.t. \sum_{i=1}^{d} w_i^T O_{ii} w_i = 1. \tag{6.19}$$

Here, the objective in (6.18) is to maximize the pairwise correlations; and the constraints in (6.19) confine the scale of transformation. Using standard Lagrange multiplier and setting the derivatives respect to w_i to zero, we obtain the following (where λ is a Lagrange multiplier):

$$\begin{bmatrix} O_{11} & O_{12} & \cdots & O_{1d} \\ O_{21} & O_{22} & \cdots & O_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ O_{d1} & O_{d2} & \cdots & O_{dd} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} = \lambda \begin{bmatrix} O_{11} & 0 & \cdots & 0 \\ 0 & O_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & O_{dd} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \tag{6.20}$$

Recall that our structural features extracted from each dimension satisfy $C_i^T C_i = I$. Thus, the matrix $diag(O_{11}, O_{22}, \ldots, O_{dd})$ in (6.20) becomes an identity matrix. Hence $\mathbf{w} = [w_1, w_2, \cdots, w_d]^T$ corresponds the top eigenvector of the full covariance matrix on the left-hand side in (6.20). This essentially equals to PCA applied to data of the following form:

$$X = [C_1, C_2, \ldots, C_d]. \tag{6.21}$$

After the transformation w to the structural feature sets, the corresponding community at each dimension get aligned with each other. In order to partition the actors into k disjoint communities, we can extract the top $k-1$ dimensions such that the community structure is most prominent. Let $X = UDV^T$ be the SVD of X. It follows that the top $(k-1)$ vectors of U are the lower dimensional embedding.

In summary, to handle multiple dimensions of network interaction, we can first extract structural features from each dimension. Then, we concatenate all the structural features and perform PCA to find out the low-dimensional embedding. Based on the embedding, k-means can be applied to find out the discrete community assignment. The detailed algorithm is summarized in Fig. 6.5.

Input: $\mathcal{R} = \{R_1, R_2, \ldots, R_d\}$, k , ℓ
Output: idx, C_i $(i = 1, 2, \ldots, d)$

1. Compute top ℓ structural features C_i based certain criteria as in Eq. (14)-(16);
2. Construct $X = [C_1, C_2, \ldots, C_d]$;
3. Compute slim SVD of $X = UDV^T$;
4. Obtain lower-dimensional embedding $\widetilde{U} = U(:, 1 : k-1)$;
5. Calculate the cluster idx with k-means on $\widetilde{U}$.

Fig. 6.5 Algorithm for community extraction in multi-dimensional networks

Different from the two alternatives (average interaction or average criteria to optimize), the proposed approach is more robust to noisy interactions in multi-dimensional networks [23]. Moreover, this scheme does not require any weighting scheme for real-world deployment.

6.3.3 Connections Between Multi-mode and Multi-dimensional Networks

Comparing the algorithms for multi-mode networks and multi-dimensional networks, we can find a common component: *extract structural features and concatenate them to form a feature-based data set of actors, and then apply SVD to obtain the lower dimensional embedding* (Steps 4 and 5 in Fig. 6.4 and Steps 2–4 in Fig. 6.5). The basic scheme is to *convert the network interactions into features*. This scheme can work not only for community identification but also for relational learning and behavior prediction [18].

A social media network can be both multi-mode and multi-dimensional. One can combine the two algorithms to handle multi-mode and multi-dimensional challenges. The combination is straightforward: if there are within-mode interactions that are multi-dimensional, we can simply append to P_i in (6.11) with some structural features that are indicative of the community structure. That is,

$$P_i = \left[\left\{ \sqrt{w_{ij}}\, R_{i,j} C_j \right\}_{i<j}, \quad \left\{ \sqrt{w_{ki}}\, R_{k,i}^T C_k \right\}_{k<i}, \quad \{C_i^d\} \right], \tag{6.22}$$

where C_i^d denotes the structural features extracted from dth dimension of interaction in the ith mode. In this way the presented algorithm is able to handle diverse heterogeneous networks.

6.4 Understanding Groups

In earlier sections, we concentrate on group structures. That is, how to extract groups from network topology. Extracting groups is the first step for further analysis to answer questions such as *why are these people connected to each other?* and *what*

is the relationship between different groups? In this section, we seek to capture group profiles in terms of topics or interests they share [24]. This helps understand group formation as well as other group related task analysis. As the total number of groups' interests can be huge and might change over time, a static group profile cannot keep pace with an evolving environment. Therefore, *online group profiling based on topic taxonomy* [21] is proposed to serve the need.

6.4.1 Group Profiling

While a large body of work has been devoted to discover groups based on network topology, few systematically delve into the extracted groups to understand the formation of a group. Some fundamental questions remain unaddressed:

What is the particular reason that binds the group members together?
How to *interpret* and *understand* a social structure emanated from a network?

Some work attempts to understand the group formation based on statistical structural analysis. Backstrom et al. [2] studied prominent online groups in the digital domain, aiming at answering some basic questions about the evolution of groups, like *what are the structural features that influence whether individuals will join communities*. They found that the number of friends in a group is the most important factor to determine whether a new user would join the group. This result is interesting, though not surprising. It provides a global level of structural analysis to help understand how communities attract new members. However, more efforts are required to understand the formation of a particular group.

According to the concept of Homophily [14], a connection occurs at a higher rate between similar people than dissimilar people. Homophily is one of the first characteristics studied by early social science researchers and holds for a wide variety of relationships [14]. Homophily is also observed in social media [5, 26]. In order to understand the formation of a group, the *inverse* problem can be investigated: Given a group of users, can we figure out why they are connected? What are their shared similarities? *Group Profiling* [24], by extracting shared attributes of group members, is one approach proposed to address the problem.

Besides understanding social structures, group profiling also helps for network visualization and navigation. It has potential applications for event alarming, direct marketing, or group tracking. As for direct marketing, it is possible that the online consumers of products naturally form several groups, and each group posts different comments and opinions on the product. If a profile can be constructed for each group, the company can design new products accordingly based on the feedback of various groups. Group profiles can be also used to connect dots on the Web. It is noticed that an online network (e.g., blogosphere) can be divided into three regions: singletons who do not interact with others, isolated communities, and a giant connected component [11]. Isolated communities actually occupy a very stable portion of the entire network, and the likelihood for two isolated communities to merge is

very low as the network evolves. If group profiles are available, it is possible for one group or a singleton to find other similar groups and make connections of segregated groups of similar interests.

A set of topics can be used to describe a group. Since a group consists of people with shared interests, one intuitive way of group profiling is to clip a group with some topics shared by most members in the group. Luckily, social media provides not only network connectivity but also textual information. For instance, in blogosphere, bloggers upload blog posts; in content-sharing sites like Digg.com and Del.icio.us, users post news or bookmarks. These content information essentially represents the latent interests of individuals. Moreover, users also provide tags on the shared content. These tags can serve as topics.

In order to achieve effective group profiling, one straightforward approach is *aggregation*. For instance, if a tag is commonly used by the majority of group members, then the tags with highest frequency can be used to describe the group. This technique is widely used to construct tag clouds to capture the topic trend of a social media site. However, as pointed out in [24], aggregation can lead to selection of irrelevant tags for a group, especially those popular tags. This is even worse if the topics are extracted from raw text such as blog posts, comments, and status updates. Instead, to find out the description of a group, *differentiation*-based method can be exploited. That is, we can treat the group as a positive class, and the remaining actors in the network as a negative class. Then, only those features that occur frequently in the group while rarely outside the group are selected. More interestingly, it is empirically shown that by comparing the group with its neighboring actors (those actors outside the group but connecting to at least one member in the group), the extracted features are equivalently informative. Essentially, we can consider the group as a unit and take an egocentric view. The group profiles can be extracted by differentiate the group from their friends (denoted as *ego-differentiation*).

Table 6.2 shows one example of profiles extracted based on different strategies on Blythedoll group[1] of over 2000 members in a popular blog site LiveJournal.[2] Blythedoll was first created in 1972 by US toy company Kenner, later it spread out to the world. Takara, a Japanese company, is one of the most famous producers. As seen in the table, the aggregation-based method tends to select some popular interests such as *music*, *photography*, *reading*, and *cats*. On the contrary, differentiation-based methods select interests that are more descriptive. This pattern is more observable when the profiles are constructed from individual blog posts. Aggregation reports a profile that is hardly meaningful, while differentiation still works reasonably well. Even if we take an egocentric view for the differentiation-based method, a similar result is observed.

[1] http://community.livejournal.com/blythedoll/profile
[2] http://www.livejournal.com/

Table 6.2 Profiles constructed by various strategies for Blythedoll group in LiveJournal

Profiles based on individual interests		
Aggregation	Differentiation	Ego-differentiation
blythe	blythe	blythe
photography	dolls	dolls
sewing	sewing	sewing
japan	japan	blythe dolls
dolls	blythe dolls	super dollfie
cats	super dollfie	japan
art	hello kitty	hello kitty
music	knitting	toys
reading	toys	knitting
fashion	junko mizuno	re-ment
Profiles based on blog posts		
Aggregation	Differentiation	Ego-differentiation
love	blythe	blythe
back	doll	doll
ll	flickr	dolly
people	ebay	dolls
work	dolls	ebay
things	photos	sewing
thing	dolly	flickr
feel	outfit	blythes
life	sell	outfit
pretty	vintage	dollies

Each profile consists of the top 10 selected features. The first block shows the profiles constructed based on individual interests on user profiles and the second block based on group members' blog posts

6.4.2 Topic Taxonomy Adaptation

In social media, there are hundreds of thousands of online groups with diverse interests. The topics associated with different groups can be inordinate, and the total number of topics can be huge. Moreover, the selected topics in group profiles can be highly correlated as different users use tags or words at different granularity. Facing a large number of topics, we need to find a more suitable representation to understand the relationship between different groups.

Organizing the topics into a tree-structured taxonomy or hierarchy is a natural solution, as it provides more contextual information with refined granularity compared with a flat list. The left tree in Fig. 6.6 shows one simple example of a topic taxonomy. Basically, each group is associated with a list of topics. Each topic can be either a non-leaf (internal) node like *Meteorology* or *Politics*, or a leaf node like *Hurricane*. Different groups can have shared topics. Given a topic taxonomy, it is easy to find related or similar topics via parent, sibling, or child nodes. Taxonomies also facilitate the visualization of relationships between different groups and the detection of related or similar groups.

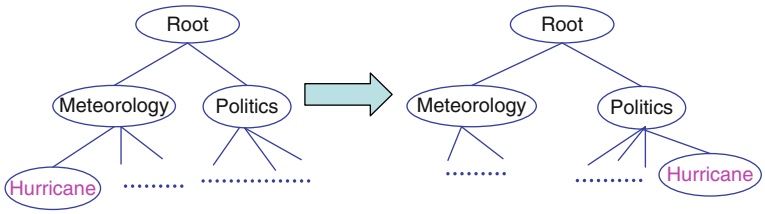


Fig. 6.6 "Hurricane" example

A topic taxonomy can be provided by human beings based on topic semantics or abridged from a very large taxonomy like Yahoo! or Google directory. It is a relatively stable description. However, group interests develop and change. Let us look at an example about "Hurricane" [25]. As shown in Fig. 6.6, in a conventional topic taxonomy, the topic *Hurricane* is likely to locate under *Meteorology* and not related to *Politics*. Suppose we have two groups: one is interested in *Meteorology* and the other in *Politics*. The two groups have their own interests. One would not expect that "Hurricane" is one of the key topics under *Politics*. However, in a period of time in 2005, there was a surge of documents/discussions on "Hurricane" under *Politics*. Before we delve into why this happened, this example suggests *the change of group interests and the need for corresponding change of the taxonomy*. This reason for this shift is that, a good number of online documents in topic *Hurricane* are more about *Politics* because Hurricanes "Katrina" and "Rita" in the United States in 2005 caused unprecedented damages to life and properties; and some of the damages might be due to the faults of federal emergency management agency in preparation for and responding to the disasters.

This example above demonstrates some inconsistency between a stagnant taxonomy and changing interests of an online group. Group interests might shift and the semantics of a topic could be changed due to a recent event. To enable a topic taxonomy to profile the changing group interest, we need to allow the topic taxonomy to adapt accordingly and reflect the change. The dynamic changes of semantics are reflected in documents under each topic, just like in the *hurricane example*. This observation motivates us to adjust a given topic taxonomy in a data-driven fashion.

Figure 6.7 illustrates a typical process of topic taxonomy adaption. By observing the difference between the original taxonomy and the newly generated taxonomy, we notice that topics can emerge and disappear for various groups. Given recent text data (e.g., tags, blog posts, visited web pages, submitted search queries) extracted from social media and a given topic taxonomy, we aim to automatically find a revised taxonomy of topics (tags) that is consistent with the data and captures dynamic group interests.

One fundamental question is how to measure the discrepancy between the semantics reflected in textual contents and a topic taxonomy. While it is a thorny challenge to quantify the discrepancy, a surrogate measure, the classification performance based on the topic taxonomy can be calibrated. In order to obtain the classification performance, we can exploit the content and tag information from social media. *The*

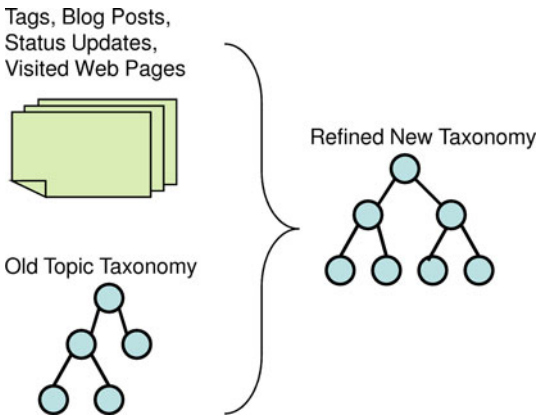


Fig. 6.7 Topic taxonomy adaptation

tags provide topic information while the shared contents act like data. With a robust hierarchical classifier built from some collected data and an existent taxonomy, new documents can be labeled automatically by the classifier. If the label are consistent with the associated tags, the taxonomy, in a sense, captures the relationship of tags. So the corresponding classification performance based on a taxonomy is one effective way of indirectly measuring how good a topic taxonomy is to represent relationships of different topics. In other words, the quality of a topic taxonomy reduces to the classification performance (e.g. recall, precision, ROC) based on the taxonomy.

We can change the topic taxonomy via classification learning as shown in Fig. 6.8. Suppose a topic taxonomy is constructed based on text information from before. The taxonomy is then adapted to maximize the classification performance on the newly arrived texts. The basic idea is, given a predefined taxonomy, a different hierarchy can be obtained by performing certain operations. Then, the newly generated hierarchy is evaluated on collected shared contents with tag information. If the taxonomy change results in a performance improvement, it is kept; otherwise, alternative change to the original taxonomy is explored. This process is repeated until no more change can lead to performance improvement, ending up with a new taxonomy which acclimatizes the taxonomic semantics according to the contents.

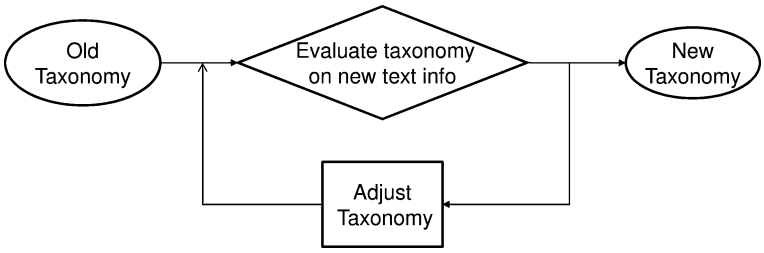


Fig. 6.8 Taxonomy adaptation via classification learning

Since a topic taxonomy does not change considerably in a short time period, we expect only a small portion of tags change their positions in the taxonomy. Tang et al. [21, 25] propose to adapt a provided taxonomy *locally* according the classification performance on novel data. Three elementary operations are defined to change a taxonomy locally as shown in Fig. 6.9:

– Promote: roll up one node to upper level;
– Demote: push down one node to its sibling;
– Merge: merge two sibling nodes to form a super node;

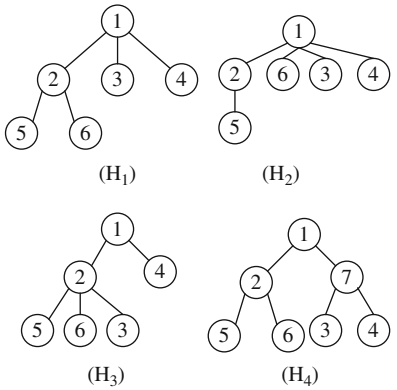


Fig. 6.9 Elementary operations. H_1 is the original hierarchy. H_2, H_3, and H_4 are obtained by performing different elementary operations. H_2: promote node 6; H_3: demote node 3 under node 2; and H_4: merge node 3 and node 4

Since the defined local changes can be applied to any node in a taxonomy, the total number of operations can be abundant. Generally, the nodes at higher level play a more important role for classification. Hence, it is proposed to follow a top-down traversal of a hierarchy to search for applicable operations [25]. It is empirically shown that two iterations of the traversal are often sufficient to achieve a robust taxonomy that captures the dynamic relationship between different groups.

6.5 Summary and Future Work

Social media is replete with diverse and unique information. It provides heterogeneous network data as well as collective wisdom in forms of user-generated contents and tags. In this chapter, we present important research tasks and intriguing challenges with social media and elaborate issues related to the understanding of online group structures and properties. In particular, we discuss two aspects of the problem: (1) how to extract communities given multi-mode and multi-dimensional data and (2) how to dynamically capture group profiles and relationships.

The social media networks are heterogeneous: their idiosyncratic entities and various interactions within the same network result in multi-mode and multi-dimensional networks, respectively. Although abundant, the information can be sparse, noisy, and partial. Therefore, special care is required to understand group structures and properties. We present some feasible solutions to extract reliable community structures in both types of networks. We also show that the two algorithms share a common component to extract "structural features" from each mode or dimension and then concatenate them to find some lower dimensional embedding which is indicative of some community structure. This simple scheme has been shown effective in community extraction in social media. Another task equally important to community extraction is to capture group interests based on textual and tag information. We describe strategies to perform effective group profiling, as well as topic taxonomy adaptation to capture dynamic group relationship using noisy and time-sensitive tag and content information.

This chapter has only addressed a couple of essential issues. Many research directions are worthy pursuing in our endeavor to understand group structures and properties in social media. We propose the following for further research:

- How can one determine the number of communities in heterogeneous networks? In the current models, we assume the number of communities at each mode or dimension is fixed. Some parameter-free process will be very useful to automatically determine the number of communities.
- It is interesting to study communities at different degrees of granularity in heterogeneous networks. One possibility is to handle heterogeneity with hierarchical clustering.
- To deal with multi-dimensional networks, our current solution is to integrate different dimensions of interactions globally. Since it is more likely that some groups are more involved in one dimension than in other dimensions, can we integrate the interactions in different dimensions differently depending on dimensional intensities? It is a challenge to simultaneously discover a common community structure as well as the integration scheme for each group.
- Extracting communities in dynamic heterogeneous networks demands for effective solutions. Social media is evolving continuously, newcomers joining the network, extant members generating new connections or becoming dormant. It is imperative to efficiently update the acquired community structure. It is also interesting to consider the temporal change of individuals for community detection.
- The work of group profiling only employs descriptive tags and contents to profile groups. More can be attempted for group profiling. For example, How to integrate the differentiation-based profiling into a taxonomy? Though the current taxonomy representation of topics does not allow one topic to have multiple parent nodes (topics), tags (especially those words with multiple meanings) can relate to different parent nodes depending on the context.
- The current scheme of group profiling is separated from group detection. If the associated tags and contents could be considered as one mode, it may be possible to exploit the methods developed for multi-mode networks to handle joint group detection and profiling.

In a nutshell, social media is a rich data source of large quantity and high variety. It is a fruitful field with many great challenges for data mining. In achieving the understanding of group structures and properties in social media, we genuinely expect that this line of research will help identify many novel problems as well as new solutions in understanding social media.

Acknowledgments This work is, in part, supported by ONR and AFOSR.

References

1. N. Agarwal, H. Liu, L. Tang, and P.S. Yu. Identifying the influential bloggers in a community. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*. Pages 207–218. ACM, New York, NY, 2008.
2. L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. Pages 44–54. ACM, New York, NY, 2006.
3. D. Chakrabarti, and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computer Survey*, 38(1): 2, 2006.
4. S. Džeroski. Multi-relational data mining: an introduction. *SIGKDD Explorations Newsletter*, 5(1): 1–16, 2003.
5. A.T. Fiore, and J.S. Donath. Homophily in online dating: When do you like someone like yourself? In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*. Pages 1371–1374. ACM, New York, NY, 2005.
6. D.R. Hardoon, S.R. Szedmak, and J.R. Shawe-taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural Computer*, 16(12): 2639–2664, 2004.
7. J. Hopcroft, O. Khan, B. Kulis, and B. Selman. Natural communities in large linked networks. In *KDD '03: Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining*. Pages 541–546. ACM, New York, NY, 2003.
8. H. Kang, L. Getoor, and L. Singh. Visual analysis of dynamic group membership in temporal social networks. *SIGKDD Explorations, Special Issue on Visual Analytics*, 9(2): 13–21, dec 2007.
9. J. Kettenring. Canonical analysis of several sets of variables. *Biometrika*, 58: 433–451, 1971.
10. J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5): 604–632, 1999.
11. R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. Pages 611–617. ACM, New York, NY, 2006.
12. B. Long, Z.M. Zhang, X. Wú, and P.S. Yu. Spectral clustering for multi-type relational data. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*. Pages 585–592. ACM, New York, NY, 2006.
13. U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, 2007.
14. M. McPherson, L. Smith-Lovin, and J.M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27: 415–444, 2001.
15. M. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E* (Statistical, Nonlinear, and Soft Matter Physics), 74(3), 2006, http://dx.doi.org/10.1103/PhysRevE.74.036104
16. A. Nielsen. Multiset canonical correlations analysis and multispectral, truly multitemporal remote sensing data. *Image Processing, IEEE Transactions on*, 11(3): 293–305, Mar 2002.

17. K. Nowicki, and T.A.B. Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American Statistical Association*, 96(455): 1077–1087, 2001.
18. L. Tang, and H. Liu. Relational learning via latent social dimensions. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. Pages 817–826. ACM, New York, NY, 2009.
19. L. Tang, and H. Liu. Scalable learning of collective behavior based on sparse social dimensions. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*. Pages 1107–1116. ACM, New York, NY, 2009.
20. L. Tang, and H. Liu. Uncovering cross-dimension group structures in multi-dimensional networks. In *SDM workshop on Analysis of Dynamic Networks*, Sparks, NV, 2009.
21. L. Tang, H. Liu, J. Zhang, N. Agarwal, and J.J. Salerno. Topic taxonomy adaptation for group profiling. *ACM Transactions on Knowledge Discovery from Data*, 1(4): 1–28, 2008.
22. L. Tang, H. Liu, J. Zhang, and Z. Nazeri. Community evolution in dynamic multi-mode networks. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. Pages 677–685. ACM, New York, NY, 2008.
23. L. Tang, X. Wang, and H. Liu. Uncovering groups via heterogeneous interaction analysis. In *Proceeding of IEEE International Conference on Data Mining*. Pages 503–512, Miami, FL, 2009.
24. L. Tang, X. Wang, and H. Liu. Understanding emerging social strucutres: A group-profiling approach. *Technical report*, Arizona State University, 2010.
25. L. Tang, J. Zhang, and H. Liu. Acclimatizing taxonomic semantics for hierarchical content classification. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. Pages 384–393. ACM, New York, NY, 2006.
26. M. Thelwall. Homophily in myspace. *Journal of the American Society for Information Science and Technology*, 60(2): 219–231, 2009.
27. S. Wasserman, and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press Cambridge, 1994.
28. K. Yu, S. Yu, and V. Tresp. Soft clsutering on graphs. In *NIPS*, Vancouver, Canada, 2005.
29. D. Zhou, and C.J.C. Burges. Spectral clustering and transductive learning with multiple views. In *ICML '07: Proceedings of the 24th international conference on Machine learning*. Pages 1159–1166. ACM, New York, NY, 2007.

Chapter 7
Time Sensitive Ranking with Application to Publication Search

Xin Li, Bing Liu, and Philip S. Yu

Abstract Link-based ranking has contributed significantly to the success of Web search. PageRank and HITS are the most well-known link-based ranking algorithms. These algorithms are motivated by the observation that a hyperlink from a page to another is an implicit conveyance of authority to the target page. However, these algorithms do not consider an important dimension of search, the temporal dimension. These techniques favor older pages because these pages have many in-links accumulated over time. New pages, which may be of high quality, have few or no in-links and are left behind. Research publication search has the same problem. This project investigates the temporal aspect of search in the framework of PageRank with application to publication search. Existing remedies to PageRank are mostly heuristic approaches. This project proposes a principled method based on the stationary probability distribution of the Markov chain. The new algorithm, TS-Rank (for *Time Sensitive Rank*), generalizes PageRank. Methods are also presented to rank new papers that have few or no citations. The proposed methods are evaluated empirically; the results show the proposed methods are highly effective.

7.1 Introduction

The main task of search engines is to find the most relevant and quality pages given a user query that reflects the user's information needs. The most successful techniques are those that exploit the social forces of people who present information on the Web [22]. Two most well-known techniques are PageRank [11] and HITS [28]. These techniques are motivated by the observation that a hyperlink (or simply link for short) from a Web page to another is an implicit conveyance of authority to the target page. Thus, a page with more in-links (links pointing to the page) is in general of higher quality than a page with fewer in-links. These algorithms are used to find quality pages and to rank the pages according to their quality scores.

B. Liu (✉)
Department of Computer Science, University of Illinois at Chicago, 851 S. Morgan (M/C 152), Chicago, IL 60607-7053, USA
e-mail: liub@cs.uic.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_7,

However, an important aspect that is not considered by these classic techniques is the timeliness of search results. The Web is a dynamic environment. It changes constantly. Quality pages in the past may not be quality pages now or in the future. In this project, we study the temporal aspect of search, which is important because users are often interested in the latest information. Apart from well-established facts and classics, which do not change much over time, most contents on the Web change constantly. New pages or contents are added. (Ideally) Outdated contents and pages are deleted. However, in practice many outdated pages and links are not deleted. This causes problems for search engines because such outdated pages can still be ranked high due to the fact that they have existed on the Web for a long time and have accumulated many in-links. Those high-quality new pages with the most up-to-date information will be ranked low because they have few or no in-links. It is thus difficult for users to find the latest information on the Web based on the current search technology. The problem is almost the same for publication search except that research publications and their reference lists cannot be deleted after publication.

We believe that dealing with the temporal dimension of search is of great importance to the development of future search technologies. Although it is possible that current search engines have already considered the time in their ranking algorithms (but kept secret), it is still important to thoroughly investigate the issue openly. Such studies will enable both the research and industrial communities to have a better understanding of the problems and to produce effective and principled solutions. The resulting algorithms will help both future and current search engines. Recently, several researchers have started to address this problem [4, 17, 36, 39]. We will discuss them in the next section.

To understand the temporal issues better, let us analyze different kinds of Web pages. We can coarsely classify Web pages into two types, old pages and new pages for simplicity of explanation. Similarly, from the dimension of reputation/quality, we coarsely divide the pages into quality pages and common pages. Roughly speaking, quality pages are those pages that have a large number of in-links, i.e., perceived by users to have authoritative contents. Common pages are those that do not have many in-links.

Old pages: These are the pages that have appeared on the Web for a long time. Let us first discuss those old quality pages, which can be further classified based on the temporal dimension:

1. Old quality pages that are up to date: As the time goes by the authors of these pages update their contents to reflect the latest developments. Such pages often stay as quality pages, which are indicated by the fact that they keep receiving new in-links over time (as more users and new generations of users are interested in the topics). These pages are still valuable. PageRank is able to give them high rank scores.
2. Old quality pages that are not up to date: These pages become outdated and no longer represent the state of the art. They become common pages, which are reflected by the fact that they receive fewer and fewer new in-links over time, and some old links may also be deleted. However, if many Web users do not clean

up their pages to delete outdated links, which are often the case, such pages can still maintain sizeable sets of in-links. Then, they will still be ranked high by PageRank although they may have little value at the present time.

Regarding old common pages, we can analyze them similarly:

1. Old common pages that remain common: Most pages on the Web are such pages. Over time, they do not receive many in-links. They do not cause problem for PageRank.
2. Old common pages that have become important: These pages were not important in the past but as time goes by they become valuable pages. This transition can be due to a number of reasons, such as fashion change, or quality contents being added by the authors. Over time such pages receive more and more in-links. PageRank is able to rank them high.

New pages: These are pages that appear on the Web recently. In general, they are ranked low by PageRank because they have fewer or no in-links. However, some of these pages may be of high quality, but PageRank is unable to rank them high.

In summary, for a ranking algorithm to consider the temporal dimension of search, two problems need to be dealt with in page evaluation:

1. How to assign a lower importance score to an old quality page that is not up to date or out of favor, but still has a sizeable set of old in-links.
2. How to assign a higher importance value to a new quality page that has few or no in-links.

Both these cases present difficulties to the PageRank algorithm. In this project, we attempt to deal with these problems. The key is to take time into consideration in evaluating the quality of a page.

We investigate these problems in the context of research publication search due to several reasons:

1. Results in the research publication domain can be objectively evaluated as we can count the number of citations received by a paper in the "future" (represented by test data) to see whether our evaluation is appropriate at the present time. Future citation count of a paper is a commonly used indicator of quality and impact of a research paper. Given a collection of papers and journals, all the information required in evaluation is readily available. In contrast, on the Web, without a search engine to constantly crawl the Web it is hard to know when a particular hyperlink was installed, and when a page is created and published on the Web. Unfortunately we do not have facilities for such crawling.
2. Concepts and entities in both domains are quite similar. Their effects and functions in the two domains are also comparable. For example, a research paper corresponds to a Web page, and a citation to a research paper corresponds to a hyperlink to a Web page. We will discuss more similarities and also some differences later in Section 7.6.

3. Publication search is important and useful in its own right. With the popularity of digital libraries on the Web, the ability to search for relevant and quality publications is valuable for both research and education.

In this project, we perform a focused study of the citation-based evaluation of research papers, which corresponds to the hyperlink-based evaluation of Web pages. The publication time of each paper is explicitly integrated into the ranking model. Although the time factor has been studied by several researchers, existing formulations are mostly heuristic modifications of PageRank. This project proposes a principled algorithm based on stationary probability distributions of Markov chains. The new algorithm, called TS-Rank (for Time Sensitive Rank), generalizes PageRank. If time is not considered in the algorithm, it reduces to PageRank. Source evaluations are also studied to rank papers that have few or no in-links. The proposed technique is evaluated empirically using a large publication collection of high-energy particle physics of 9 years. The results show that the proposed method is highly effective and outperforms the recently proposed method.

7.2 Related Work

Since PageRank [11] and HITS [28] were published, a large number of papers on improvements, variations, and speed-up of the algorithms have appeared in the literature [1, 2, 6, 7, 10, 12, 15, 20–22, 27, 30, 32, 33, 35, 38]. Many applications of the algorithms have also been reported, in both Web search and research publication search, e.g., search engines, Web resource discovery [7, 13, 14], Web community mining [23, 29], adaptive search [1], search considering both hyperlinks and page contents [14, 26], and research paper search [24, 31, 34] and social network analysis [30]. These works are still within the framework of the original formulation of the algorithms and do not consider the temporal aspect. References [16, 35] study the evolution of the Web and identify the same problem as we discussed above. A number of interesting phenomena about the Web evolution are reported. However, no technique was proposed to deal with the problem.

In recent years, several researchers have tried to deal with the temporal dimension of search and to study ways to promote new pages [4, 17, 36, 39]. Reference [36] proposes a randomized ranking method to randomly promote some new pages so that they can accumulate links quickly. Reference [17] uses the derivatives of PageRank to forecast future PageRank values for new pages. These approaches are quite different from ours, as we deal with both new and old pages and propose a principled method that integrates time naturally within the ranking algorithm.

The most closely related works to ours are those in [4] and [39]. They both consider time in the ranking algorithm directly. They make some heuristic modifications to PageRank. For example, the recent method (called TPR) given in [39], which is evaluated using research publications, attaches a weight to each PageRank score to reduce the effect of old papers. The algorithm is

$$P(x_i) = (1-d) + d \times \sum_{x_j \in IN(x_i)} \frac{w_j \times P(x_j)}{O_j}, \tag{7.1}$$

where

$P(x_i)$ is the rank score of paper x_i.
O_i is the number of out-links or references of the paper x_j, i.e., the number of references in the reference list.
$\text{IN}(x_i)$ is the set of papers that cites x_i, i.e., the in-links of x_i.
d is a damping factor, ranging between 0 and 1, which is used in the original PageRank equation.
w_j is the weight to reduce the effect of old links (or citations). It is a function of the time when the paper x_j is published. The intuition is that a citation occurred recently is more important than a citation occurred a longtime ago.

Equation (7.1) is the same as the PageRank equation in [11] except that w_j are added. The original PageRank algorithm was derived based on the Markov chain and a random surfer. The PageRank value of each page is the stationary probability that the random surfer will arrive at the page. A Markov chain is represented with a stochastic transition matrix, which, in the context of the Web search, means that at a particular page x_j (or a state in the Markov chain), if the page x_j has O_j out-links, the sum of probabilities of following each link to go to another page (or state) must be 1. This is true because PageRank gives each link a probability of $1/O_j$ (i.e., (7.1) without w_j). Equation (7.1) does not meet this requirement because due to the weight factor w_j, the probabilities of going from one page to other pages no longer sum up to 1. The sum is in fact w_j, which is between 0 and 1. In [4], three other modifications to PageRank were also suggested, but not fully evaluated. Again, the modifications were ad hoc with little theoretical foundation. This project corrects this situation and proposes a more principled approach to consider time in the ranking algorithm naturally.

On publication search, [31] describes the CiteSeer system. CiteSeer is a popular digital library on the Web for research publication search and citation analysis. CiteSeer also uses PageRank and HITS algorithm in the system. It is thus able to rank papers by either "hub" or "authority" score. Reference [31] mentions that the temporal aspect should be considered in publication search. However, the topic was not further investigated. A more recent report on CiteSeer is in [24]. In [34], a ranking method called PopRank was proposed to rank objects on the Web and was applied to publication search. However, the method does not consider time. Google scholar[1] is another publication search system, but we could not find any published work on the ranking method used in it. There are also many other publication search

[1] http://scholar.google.com

(digital library) systems such as DBLP,[2] NDLTD,[3] NCSTRL,[4] Cora,[5] and CoRR.[6] However, these systems only allow simple keyword search based on information retrieval methods and/or the PageRank and HITS algorithms.

7.3 The Proposed TS-Rank

There are many factors that contribute to the ranking of research papers or Web pages. Broadly speaking, we can group them into content-based factors and reputation-based factors.

Content-based factors: These factors are related to the contents of research publications or Web pages that the user is seeking. In the context of research publications, such factors may include how many user query words are contained in the paper and how far these words are from each other. In this research, we do not focus on these factors.

Reputation-based factors: Typically there are many relevant Web pages or research publications based on contents. Reputations of papers help to determine the ranking of the papers to be presented to the user. In the context of publication search, reputation factors include the citation count of the paper, the reputation of its authors, and the reputation of the journal or conference where the paper is published.

This work focuses on reputation-based factors and studies how the temporal dimension may be integrated into the evaluation of the reputation of a research paper.

We now switch to the domain of research publications (or papers) and use it as a case study to show how the temporal dimension can help to improve search ranking. However, most discussions below are also applicable to Web pages.

As indicated earlier, there are two main factors contributing to the reputation of a paper:

1. The number of in-links of the paper, i.e., the number of citations that the paper receives.
2. Source of the paper. There are two sources for each paper:
   - the authors of the paper and
   - the journal or the conference where the paper is published.

The reputations of these sources affect the reputation of the paper, especially when the paper is new and has few or no citations.

There are two main timing factors related to a research paper.

[2] http://www.informatik.uni-trier.de/~ley/db/
[3] http://www.ndltd.org/
[4] http://www.ncstrl.org/
[5] http://cora.whizbang.com/
[6] http://xxx.lanl.gov/archive/cs/intro.html

1. The publication date of the paper
2. The dates that the paper is cited by other papers, which are the publication dates of these other papers

These timing factors are important because the user is usually interested in the most recent research results. A paper published a longtime ago and has many citations accumulated a longtime ago may be less important than a quality paper that is published recently with fewer citations.

Among the two factors above, the second factor is of primary importance because it reflects the relevancy, importance, and timeliness of the paper as perceived by other researchers. Although a paper may be published a longtime ago, if it still receives a large number of recent citations, it is still relevant and important. However, if an old paper receives few recent citations, it is an indication that the paper has become less important now.

Below, we derive the TS-Rank algorithm to consider the time explicitly in the evaluation of the reputation of a paper.

7.3.1 The TS-Rank Algorithm

A simple way to consider time in ranking is to simply use only those in-links (citations) that are received by each paper recently (e.g., in a specific time window) in the PageRank computation. All earlier citations received by each paper are discarded. However, this approach is too crude as it may remove many older papers from consideration completely because they may receive few or even no citations in the recent time window. Although the topics of these papers may be out of fashion or few researchers are still working them, they may still be interesting to some users. When the user searches for such a topic, we still want those old important papers ranked higher than those old common papers. It is thus desirable to use a time decay function to weigh old citations less than new citations and to have this done in a principled manner. Discarding old citations completely is not appropriate. Furthermore, this simple method does not handle new papers that have few or no citations. They will still be ranked low by PageRank.

We now derive the TS-Rank method. We use the Markov chain model and a random reader to formulate the problem. The collection of all papers $\{x_1, x_2, \ldots, x_n\}$ is converted to a graph $G = (V, E)$, where V is the set of papers and E is the set of all directed links or edges (which are citations). In the Markov model, we treat G as the Markov chain, each paper as a state, and each edge as a transition from one state to another. The random reader browses and reads papers following the references in the reference list of each paper. In the Markov model, we say that the reader performs state transition. If we assume that the reader will follow each reference uniformly at random, the probability of following each reference is $1/O_i$, where O_i is the number of out-links or references of the paper. In the Markov chain, $1/O_i$ is the transition probability of moving from state to another state. Considering all the states (papers) in the Markov chain, we have a transition probability matrix of the chain, denoted by $\boldsymbol{A}$. Each cell of the matrix $\boldsymbol{A}$ is defined as the following.

$$A_{ij} = \begin{cases} \dfrac{1}{O_i} & \text{if}(i, j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{7.2}$$

Since every paper has a reference list, the sum of all transition probabilities of each state is 1, which means that $\boldsymbol{A}$ is the *stochastic transition matrix* of a Markov chain, i.e.,

$$\sum_{j=1}^{n} A_{ij} = 1, \tag{7.3}$$

where n is total number of states (papers) in the chain. Even if a paper x_i does not have a reference list, we can easily make $\boldsymbol{A}$ a stochastic matrix by assuming that x_i cites every paper in the collection. As a notational convention, we use bold and italic letters to represent matrices.

By the Ergodic theorem of Markov chains [37], a finite Markov chain defined by the *stochastic transition matrix* $\boldsymbol{A}$ has a unique *stationary probability distribution* if $\boldsymbol{A}$ is *irreducible* and *aperiodic*. The stationary probability distribution means that after a series of transitions the probability of the random reader arriving at each state will converge to a steady-state probability regardless of the choice of the initial probability at each state. With all the steady-state probabilities of the states, we have a steady-state probability distribution vector $\boldsymbol{P}$ (expressed as a column vector). According to the Markov chain model, the following equation holds at the steady state:

$$\boldsymbol{P} = \boldsymbol{A}^{\mathrm{T}} \boldsymbol{P}. \tag{7.4}$$

$\boldsymbol{P}$ is in fact the PageRank (column) vector containing all the PageRank values of all the papers (states). The superscript T means the matrix transpose. $\boldsymbol{P}$ is the *principal eigenvector* of $\boldsymbol{A}^{\mathrm{T}}$ with *eigenvalue* of 1. If we do not use matrix notation, (7.4) is

$$P(x_i) = \sum_{j=1}^{n} A_{ji} P(x_j). \tag{7.5}$$

We now consider the other two conditions, *irreducible* and *aperiodic*. Irreducibility of $\boldsymbol{A}$ means that the citation graph G is strongly connected, i.e., for any pair of vertices, u and v, there is a directed path from u to v. However, this does not hold for our citation graph because older papers will not cite new papers. Thus there is no way to transit from an older paper to a new paper.

To make $\boldsymbol{A}$ is irreducible, we can use a similar trick as in PageRank. We add an artificial link (citation) from each state to every state and give each link a small transition probability controlled by a time function $f(t)(0 \leq f(t) \leq 1)$, where t is the time difference between the current time and the time when the paper is published. $f(t)$ returns a probability that the reader will follow an actual link or

citation in the paper. $1 - f(t)$ returns the probability that the reader will jump to a random paper. Thus, at a particular paper x_i, the random reader has two options:

1. With probability $f(t_i)$, he randomly chooses a reference to follow
2. With probability $1 - f(t_i)$, he jumps to a random page without a citation

The intuition here is that if the paper is published a longtime ago, the papers that it cites are even older and are probably out of date. Then the $1 - f(t)$ value for such a paper should be large, which means that the reader will have a high probability of jumping to a random paper. If a paper is new, then its $1 - f(t)$ value should be small, which means that the reader will have a high probability to follow a reference (citation) of the paper and a small probability of jumping to a random paper.

With the above augmentation, (7.5) becomes

$$P_T(x_i) = \sum_{j=1}^{n} \left(\frac{1 - f(t_j)}{n} + f(t_j) A_{ji} \right) P_T(x_j), \tag{7.6}$$

which is the TS-Rank of page x_i denoted by P_T. $1/n$ is the probability of going to a random paper x_j. Recall n is the total number of papers. In the matrix notation, we have

$$\boldsymbol{P_T} = (\boldsymbol{F} + \boldsymbol{H})^T \boldsymbol{P_T}, \tag{7.7}$$

where $\boldsymbol{F}$ and $\boldsymbol{H}$ are both $n \times n$ square matrices defined by

$$F_{ij} = \frac{1 - f(t_i)}{n}, \tag{7.8}$$

$$H_{ij} = \begin{cases} \dfrac{f(t_i)}{O_i} & \text{if}(i, j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{7.9}$$

It is easy to show that $(\boldsymbol{F} + \boldsymbol{H})$ is a *stochastic transition matrix* of our augmented citation Markov chain. Clearly, the matrix $(\boldsymbol{F} + \boldsymbol{H})$ is also *irreducible* because due to the random jump links, the reader can go from any state to any state, i.e., the graph is strongly connected. It is also *aperiodic*. A state in a Markov chain being *periodic* essentially means that there exists a directed cycle that the chain has to traverse. If none of the states in a Markov chain is periodic, then the chain is said to be *aperiodic*. Again, due to the random jump links, the chain becomes aperiodic because there is a direct link from any state to any state and does not have to follow any fixed cycle. For formal definitions of both irreducible and aperiodic, please refer to [37].

These conditions ensure that the Markov chain (defined by $(\boldsymbol{F} + \boldsymbol{H})$ has a unique stationary probability distribution regardless of the choice of the initial probability of the random reader being at each state. The stationary probability distribution is the final TS-Rank (column) vector $\boldsymbol{P_T}$. $\boldsymbol{P_T}$ is computed as the *principal eigenvector*

of $(\boldsymbol{F} + \boldsymbol{H})^{\boldsymbol{T}}$ with *eigenvalue* of 1. To solve (7.7), the standard power iteration method can be used [37].

The final issue is how to define $f(t)$, which is application dependent. For different applications, different functions are needed depending on how the time affects the domains. For instance, in the collection of research papers of *high-energy particle physics* that we use for evaluation of this work, a new paper can receive many citations within 3–4 months and the number of citations per month can stabilize in less than 6 months because many journals in the field are published twice a month or even more frequently, and the time period from peer review to publication is very short (in terms of a few months). However, for computer science, the situation is very different. Almost all conferences are held only once a year. The publication cycle of journals from the beginning of peer review to the actual publication can take a few years. Thus, it takes at least a year (usually longer) for papers to receive sizable citations. Note that it is easy to see that if $f(t)$ is a constant between 0 and 1 for every paper, TS-Rank becomes the original PageRank [11].

For our application, we use an exponential function for $f(t)$ as exponential decay is commonly used in time series analysis. The function performs quite well in our experiments:

$$f(t) = 0.5^{t/x}, \tag{7.10}$$

where the *decay* parameter x is selected empirically. t is the difference in month between the current time and the time when the paper is published. While the effect of *decay* parameter x on the prediction results is further studied in Section 7.5.5, we use *decay* parameter $x = 3$ in the following example to illustrate the concept. For instance, in our training data (which is used to select the *decay*), the newest papers are published in December 1999. Given $x = 3$, the citations occurred in December 1999, September 1999, and June 1999 have the weights of 1, 0.5, and 0.25, respectively. The *decay* parameter can be tuned according to the nature of the data set. When its value moves toward infinitely large, the weight decreases slowly with time. It is more suitable for static domains. Similarly, if its value is close to 0, it is more suitable for highly dynamic domains.

7.3.2 Source Evaluation

Although TS-Rank considers time, it is still insufficient as it is not applicable to new papers (published recently) since they have few or no citations from other papers. To assess the potential importance of such a new paper, two pieces of source information are useful, the reputation of its authors and the reputation of the journal (or conference) where the paper is published. We make use of TS-Rank to define these two reputations.

Author evaluation: The reputation of an author is based on the research papers that he/she published in the past. We compute author evaluation by averaging the

TS-Rank values of his/her past papers. Let the papers that the author a_j has published be $x_1, x_2, \ldots, x_m$. The author score (*Author*) is computed with

$$\text{Author}(a_j) = \frac{\sum_{i=1}^{m} P_T(x_i)}{m}, \tag{7.11}$$

where $P_T(x_i)$ is the TS-Rank score of paper x_i at the present time. Due to the use of TS-Rank, this measure weighs recent citations of his/her papers more than old citations, which is reasonable as recent citations are more representative of the author's current reputation. Note that for an author who has never published a paper before, we are unable to evaluate his/her reputation.

Journal evaluation: The evaluation of each journal b_j, JournalEval (b_j), is done in the same way as that of each author by considering papers published in the journal in the past. A new journal is not evaluated.

Using the author and journal evaluations, we can estimate the importance of each new paper. However, since a paper may be co-authored by a number of people, we combine their author scores. Let the authors of the paper x_i be $a_1, a_2, \ldots, a_k$. The score of the paper based on author evaluation is given by

$$\text{AuthorEval}(x_i) = \frac{\sum_{j=1}^{k} (\text{Author}(a_j))^2}{\sum_{j=1}^{k} \text{Author}(a_j)}. \tag{7.12}$$

Clearly, there are other ways for these computations, e.g., maximum or minimum. We found that this method performs quite well compared to other alternatives.

We can also combine author evaluation and journal evaluation to score each paper. Assume that paper x_i is published in journal b_j. We can combine them by using weighted average of JournalEval (b_j) and AuthorEval(x_i):

$$AJEval(x_i) = \frac{(\text{JournalEval}(b_j))^2 + (\text{AuthorEval}(x_i))^2}{\text{JournalEval}(b_j) + \text{AuthorEval}(x_i)} \tag{7.13}$$

Note that after a paper has been published for a while, it is more effective to use TS-Rank to score the paper. Author and journal evaluations are less effective. This makes sense because after a paper has been published for some time, its citation counts reflect the impact or importance of the paper better than its authors and journal since author evaluation (or journal evaluation) is only an averaged result of all the papers of the authors (or the journal).

7.3.3 The Trend Factor

TS-Rank only assesses the value of a paper at a particular time instance based on past citations. In the time series domain, another important issue is the *trend*. If we are interested in the potential value or importance of a paper in the future, e.g.,

what is the likely importance or impact of the paper in the next year, we need to consider *trend*, which is not directly measured by TS-Rank. We now introduce the *trend factor*.

Continuing our previous example at the end of Section 7.3.1, for a paper x_i, $P_T(x_i)$ already captures the importance at the end of 1999. How does the importance change through the future year? We assume that this is reflected by the citation change at the end of 1999. Therefore, we can find the past behavior of each paper x_i to compute the *trend factor* of x_i, denoted by Trend(x_i). We define two time periods, p_1 and p_2. p_1 is the current time period (in our experiments, we use the past 3 months) and p_2 is the previous time period (i.e., the 4th, 5th, and 6th most recent months). If the papers are too young, only 2 months of data are used. Let the citation count in p_1 for paper x_i be n_1 and the citation count in p_2 for paper x_i be n_2. The trend factor of x_i is defined as

$$\text{Trend}(x_i) = n_1/n_2. \tag{7.14}$$

Considering the trend factor, paper x_i's final rank score is computed with

$$\text{Trend}(x_i) \times P_T(x_i), \tag{7.15}$$

where $P_T(x_i)$ is the TS-Rank value of paper x_i. If a paper's age is too young (e.g., less than 3 months), there is no sufficient data available to compute its trend ratio. We only use source evaluation results to evaluate the paper.

7.4 Linear Regression

For comparison purposes, we also implemented a linear regression method. The citation counts of each paper received in the latest time periods are used to perform a linear regression to predict its citation count in the next time period. This predicted citation count is used as the score of the paper. This is a reasonable method because the predicted citation count reflects the predicted impact of the paper in the next time period. The method is fairly straightforward and will not be discussed further.

As in TS-Rank, for new papers with few or no citations, we use author and journal evaluations, which can be done by using actual citation counts of all papers of the author or the journal in this case. Let the papers published by an author (a_j) be $x_1, x_2, \ldots, x_m$. Author score is computed with

$$\text{Author}(a_j) = \frac{\sum_{i=1}^{m} \text{count}(x_i)}{m} \tag{7.16}$$

where *count*(x_i) is the citation count of paper x_i. The score of a paper based on author evaluation is again given by (7.12). Journal evaluation can be done in the same way. After they are computed, (7.13) is applied to combine their scores.

It is important to note that the linear regression method based on raw citation counts is not suitable for Web search due to link spamming, which is not a major problem for research papers.

7.5 Empirical Evaluation

In this section, we evaluate the proposed techniques and compare them with PageRank and the existing method TPR in [39]. We use the KDD CUP 2003 research publication data, which are also used in [39]. This data set is from an archive of High Energy Particle Physics publications catalogued by Stanford Linear Accelerator Center.

7.5.1 Experimental Settings

Our experiments use the standard search paradigm. That is, given a collection of research papers and a user query, the system ranks the papers that are relevant to the query in the collection and presents to the user. For the purpose of this research, we assume that there is an abstract procedure that is able to determine whether a paper is relevant to a query (which is expressed as a set of keywords). In other words, our research focuses on citations and investigates the effect of time on the citation-based ranking, which is the key component of Web search (recall citations in the Web context are hyperlinks). This work does not study content-based factors such as keyword locations, their distances in the paper. We simply assume that a paper is relevant to a query if it contains all the query words.

Evaluation method: To evaluate the proposed techniques, we do not compare their rankings directly, which is harder to quantify. Instead, we compare the number of citations that the top ranking papers receive in the following year, namely, 1 year after the user performs the search. This is an objective measure. It is also reasonable because to a large extent the citation count of a paper reflects the importance of the paper. If those highly cited papers in the future are ranked high by an algorithm, it indicates that the algorithm is effective in giving users high-quality papers.

7.5.2 Experimental Results with All Papers

In this set of experiments, we use all the papers in the first 8 years (1992–1999) to perform various evaluations for the proposed methods. Following the setting in [39], we used the same 25 randomly selected queries as in [39] and rank the relevant papers of each query using the evaluation results. All query keywords were randomly selected from the set of frequent words found in the abstracts of the papers (after stopwords have been removed). The data of year 2000 are used to test various ranking methods.

Table 7.1 presents the experiment results. Only the results for the top 30 papers are given. The reason for using only top 30 ranked papers is that users seldom have the patience to look at more than even 20 papers. This is especially true for Web search.

The experimental results are presented in three rows. Each row gives the total citation counts of different methods for a group of papers. The first row is for the top 10 papers (we also call it a group of papers), where the citation count is the sum of the citation counts of all the top ten papers over the 25 queries. Similarly, the second row is for the top 20 papers, and so on. Below, we explain the results column by column.

Column 1: It lists each group of top-ranked papers.

Columns 2 and 3: Column 2 gives the result for each group of top papers based on rankings using the original PageRank algorithm, i.e., time is not considered but trend factor is used (without the trend factor, the results are much worse). Each result here is the total number of citations of each group of top-ranked papers for the 25 queries. Each count is obtained from citations that the paper receives in year 2000.

Column 3 gives the ratio of the total citation count for this method and the total citation count of the ideal ranking (called *best citation count* given in Column 14), expressed as a percentage. The ideal ranking is that one that ranks relevant papers (to a query) based on the actual number of citations received by each paper in the following year.

Columns 4 and 5: Column 4 gives the total citation count results of TPR (the method in [39] with the combined author and journal evaluations, and trend factor). Column 5 gives the same ratio as in Column 3.

Columns 6 and 7: Column 6 gives the results (total citation counts) of the TS-Rank method (with trend factor considered). Column 7 gives the same ratio as in Column 3 (the ratio of the total citation count for TS-Rank and the total citation count of the ideal ranking in Column 14). From Columns 6 and 7, we observe that TS-Rank's results are significantly better than those of the original PageRank algorithm.

Columns 8 and 9: Column 8 gives the results of TS-Rank combined with both author and journal evaluations (AJEval). Column 9 gives the same ratio as in column 7. The AJEval is only used when a paper is very new, i.e., with few or no citations. In this case, we cannot use TS-Rank. Papers are regarded as new (or very recent) if they were published less than 3 months ago. Three months are chosen because there is no sufficient data available to compute the *trend factor* for a paper younger than 3 months.

We did not list the results of author evaluation and journal evaluation individually due to space limitations in the table. They perform slightly worse than the combined method (see also Table 7.3).

From Columns 8 and 9, we can see that TS-Rank (AJEval) performs clearly better than TS-Rank alone. This is because TS-Rank could not handle new papers well.

Table 7.1 Comparison results of different methods using all papers

1	2	3	4	5	6	7	8	9	10	11	12	13	14
No. of top papers	PageRank		TPR (AJEval)		TS-Rank		**TS-Rank (AJEval)**		LR		LR(AJEval)		Best Citation Counts
10	2776	49%	4382	77%	4174	74%	**4629**	**82%**	4219	75%	4136	73%	5661
20	3720	51%	5756	78%	5719	78%	**5946**	**81%**	5371	73%	5447	74%	7345
30	4614	55%	6788	81%	6901	82%	**7154**	**85%**	6385	76%	6519	78%	8406

From Columns 4 and 5, we observe that TS-Rank outperforms the heuristic method TPR for every group of papers.

Columns 10–13 give the corresponding results of linear regression (denoted as LR in the table). After trying various possibilities, we found that using 2 years of data to build LR models gives the best results. When a paper is younger than 2 years, those old months will be treated as having 0 citation. We can see that linear regression performs reasonably well too, but worse than TS-Rank-based methods.

Column 14: It gives the best citation count for each group of paper based on the ideal ranking, i.e., ranking relevant papers based on the actual number of citations received by each paper in year 2000.

To summarize, both TS-Rank and linear regression perform significantly better than the original PageRank algorithm. The author and journal evaluations help improve the prediction results further. TS-Rank not only is a more principled method but also outperforms the heuristic method TPR (including author and journal evaluations and the trend factor). Among all the four methods, TS-Rank with author and journal evaluations gives the best result for every group of papers.

7.5.3 Results of Top 10 Papers

To give some indication of the effectiveness of ranking of the proposed methods, we find the top 10 most cited papers in 2000. We then use the proposed methods to rank all the papers that appeared from 1992 to 1999. Table 7.2 shows the ranking results.

Column 1: It shows the ranks of the top 10 papers in 2000.

Column 2: It gives the paper ID of each paper.

Column 3: It gives the rank of each paper using the original PageRank algorithm. Clearly, the results are very poor.

Column 4: It gives the rank of each paper based on the existing TPR method.

Table 7.2 Ranks of the top 10 papers

Rank	Paper ID	PageRank	TPR (AJEval)	**TS-Rank (AJEval)**	LR (AJEval)
1	9711200	19	1	**1**	1
2	9908142	742	8	**2**	5
3	9906064	613	6	**6**	10
4	9802150	39	2	**3**	2
5	9802109	46	4	**4**	3
6	9711162	323	11	**5**	7
7	9905111	576	9	**8**	4
8	9711165	620	20	**7**	14
9	9610043	17	12	**14**	19
10	9510017	7	13	**9**	8

Column 5: It gives the rank of each paper based on TS-Rank. The new papers are ranked using the combined author and journal evaluations. We see that TS-Rank again clearly outperforms the TPR method.
Column 6: It gives the rank of each paper based on linear regression. The new papers are ranked using the combined author and journal evaluations (Section 7.4).

Table 7.2 clearly demonstrates that the ranking results of the PageRank algorithm are very poor. In contrast, our proposed methods perform remarkably well. The set of predicted top eight papers are the same as those in the actual rank, and all the top 10 papers are ranked very high (within top 14). Our method also performs better than the TPR method.

7.5.4 Results on New Papers Only

In this set of experiments, we use only the new papers. That is, we only use those papers that are published less than 3 months ago from the query time. The purpose here is to assess the effectiveness of author and journal evaluations. Their results cannot be seen clearly in Table 7.1 because it includes both old and new papers, and older papers dominate in number.

This set of experiments does not directly use TS-Rank and linear regression because these papers have few or no citations. Note also that we do not use queries here because each query returns only a few results (papers) as the number of new papers is small. We use the proposed methods to rank all the new papers (i.e., all papers are considered relevant) and compare the predicted rank with the actual rank position in 2000 of the new papers.

To measure the distance between ranks, we use the Spearman footrule distance [19]. The Spearman footrule distance is the sum of the absolute difference between a paper's positions in two ranks. In our experiment, we examine the Spearman footrule distance for the top 30 papers, as users only pay attention to the top-ranked papers. Given two ranks R_1 and R_2 of size m, n papers are of our interest and $n << m$, the equation of normalized Spearman footrule distance is

$$F(R_1, R_2) = \frac{\sum_{i=1}^{n} |R_1(i) - R_2(i)|}{m \times n} \tag{7.17}$$

where $R_1(i)$ and $R_2(i)$ are the rank positions of paper i in rank R_1 and R_2

Column 1: It shows that we use Spearman footrule (SF) distance to evaluate the source evaluation.
Column 2: It shows the SF distance for the top 30 new papers between the actual rank and the rank predicted by the original PageRank.
Columns 3 and 4: Column 3 gives the SF distance of the TPR method without source evaluation. Column 4 gives the SF Equations distance of the TPR method with the combined author and journal evaluation.

Column 5: It lists the SF distance for the top 30 new papers between the actual rank and the rank predicted by TS-Rank with no source evaluation. While TS-Rank tends to underestimate the new papers, the distance value of 0.0934 indicates that new papers already collected some citations. Our decay function also helps these papers climb on the rank quickly. Therefore, using TS-Rank alone, the ranks of top new papers are already quite close to their actual ranks. We also see that TS-Rank outperforms PageRank dramatically and is also better than TPR with no source evaluation.

Columns 6 and 7: They list the SF distances of TS-Rank with author and journal evaluations, respectively. The journal distance is smaller than the author distance, which suggests that journal is a better indicator of a paper's quality, which is quite intuitive.

Column 8: It lists the SF distance results of TS-Rank with the combined author and journal evaluations. We can see that this method gives the best results on new papers. It clearly outperforms TPR when the source evaluation is applied.

Columns 9 and 10: They list the SF distances of linear regression without and with source evaluation, respectively. The results are much worse than those of TS-Rank and TPR.

The source evaluation does not help improve the ranking. An explanation is that the source evaluation over-boosted some new papers, which lowers the quality of overall ranking. A similar ranking deterioration was also observed from the top 10 group in columns 10–13 in Table 7.1. However, as we consider more papers, it shows that the source evaluation in linear regression does improve the overall ranking.

In summary, we can see from Table 7.3 that our new method in column 8 outperforms all other methods significantly.

Execution time: The time complexity of TS-Rank is the same as PageRank. However, TS-Rank takes fewer iterations to converge because the timed weight $f(t)$ on the citation links drops rapidly with the citation age. The smaller the timed weight, the less the authority can be transmitted through the citation links. Thus, timed weights restrain the accumulation of TS-Rank values. Consequently, fewer iterations are needed to converge. For our problem, PageRank converges in 36 iterations, while TS-Rank converges in 23 iterations. In each iteration, TS-Rank takes more time due to the first matrix $\boldsymbol{F}$ (the second matrix $\boldsymbol{H}$ is similar to that in PageRank). Since the time is discretized into months (papers appeared in the same month have the same $f(t)$ value), the effect of $\boldsymbol{F}$ on the computation is not large. The overall execution times of PageRank and TS-Rank are similar.

7.5.5 Sensitivity Analysis

In the introduction of the TS-Rank concept, we pointed out that decay parameter is tunable for a given data set to reach an optimal result. Our experiments show that

Table 7.3 Comparison results of different source evaluation methods using only new papers

Source evaluation	PageRank	TPR no source Eval	TPR (AJEval)	TS-Rank no source Eval	TS-Rank (Author Eval)	TS-Rank (Journal Eval)	**TS-Rank (AJEval)**	LR no source Eval	LR (AJEval)
Spearman footrule distance for top 30 new papers	0.6080	0.119	0.0565	0.0934	0.0408	0.0387	**0.0346**	0.126	0.175

TS-Rank(AJEval) is an effective scoring technique. Therefore, we apply a range of *decay* parameter values in TS-Rank(AJEval) and study the relationship between the scoring effectiveness and *decay* parameter. A set of values {1, 2, 3, 6, 12} for the decay parameter $x$is experimented. A larger decay parameter corresponds to a slow decay function. The experiment results are listed in Table 7.4.

Column 1: It lists each group of top-ranked papers.
Columns 2 and 3: Column 2 gives the results (citation counts) of the TS-Rank(AJEval) method with the decay parameter $x = 1$.
Column 3 gives the ratio of the total citation count for the TS-Rank(AJEval) method (with $x = 1$) and the total citation count of the ideal ranking (column 12), expressed as a percentage.
Columns 4–5, 6–7, 8–9, 10–11 have the similar meanings as columns 2–3. The only difference is that decay parameter varies from 2 to 12 in these experiments.
Columns 12 lists the same data showed in Column 14 of Table 7.1. It gives the best citation count for each group of papers based on the ideal ranking, i.e., ranking relevant papers based on the actual number of citations received by each paper in year 2000.

The results indicate that $x = 3$ is the optimal value for *decay* parameter in this project collection. When *decay* parameter is smaller than 3, the system heavily focuses on very recent citations. The consequence is that papers with less recent citations are absent from the predicted top papers even if they might be important. Failing to include these papers in the results lowers the overall ranking quality. On the contrary, when *decay* parameter is larger than 3, the decay function became less aggressive. Older quality papers that are not up to date will be favored because of their longer history. As a result, some new quality papers will be excluded from the top rank.

7.6 Discussions and Conclusions

This project studies the temporal dimension of search. It proposed a new algorithm in the context of publication search. Empirical evaluation verified its superior performance to existing methods. To conclude, we also discuss how TS-Rank may be applied to the general Web search.

From publication search to Web search: As indicated earlier, most concepts in research publications are parallel to the concepts in Web pages. Research papers correspond to Web pages, and journals correspond to Web sites. The date when a paper is published is the same as the date when a Web page is created or updated (most recently). The references of a paper are the same as out-links of a page on the Web. There are of course also some differences between Web pages and research papers. For example, a Web page may be deleted, but a published paper cannot be

Table 7.4 Comparison results of the TS-Rank(AJEval) technique using different *decay* parameters x

1	2	3	4	5	6	7	8	9	10	11	12
No. of top papers	Decay rate = $0.5^{t}(x=1)$		Decay rate = $0.5^{t/2}(x=2)$		Decay rate = $0.5^{t/3}(x=3)$		Decay rate = $0.5^{t/6}(x=6)$		Decay rate = $0.5^{t/12}(x=12)$		Best citation counts
10	4349	77%	4430	78%	4629	82%	4453	79%	4436	78%	5661
20	5876	80%	5948	81%	5946	81%	5924	81%	5765	78%	7345
30	7018	83%	7081	84%	7154	85%	7022	84%	6860	82%	8406

deleted. Hyperlinks can also be added to and deleted from a Web page any time, while for a research paper once published no reference or citation can be deleted or added. To consider addition and deletion in Web pages in TS-Rank, we can take one of the following two strategies:

Use the date of the most recent update to the page x_j as the creation date of the page (i.e., t_i in (7.6)), although the page may be created much earlier. This is reasonable as we can assume that the page owner updates all the links on the page. We can then directly apply (7.6).

Give links appearing in the same page different transition probabilities according to the times when they were added to the page, rather than assigning them the uniform probability of $1/O_i$. t_j is still the date when the page x_j is most recently updated. Those deleted links will not be considered any more.

All the information required for TS-Rank computation can be easily collected during Web crawling. Thus adapting TS-Rank to the Web search is fairly straightforward. In our future work, we plan to investigate and experiment with this adaptation.

Acknowledgments We thank KDD Cup 2003 organizers for making the publications and citation data available on the Web.

References

1. S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-Line page importance computation. In *WWW-2003*.
2. D. Achlioptas, A. Fiat, A. Karlin, and F. McSherry. Web search via hub synthesis. In *FOCS-2001*.
3. A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1(1): 2–43, 2001.
4. R. Baeza-Yates, F. Saint-Jean, and C. Castillo. Web dynamics, age and page quality. In S*PIRE-2002*.
5. Z. Bar-Yossef, A. Z. Broder, R. Kumar, and A. Tomkins. Sic transit gloria telae: Towards an understanding of the Web's decay, Pages 328–337, *WWW-2004*.
6. K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public Web search engines. *Computer Networks and ISDN Systems*, 30: 379–388, 1998.
7. K. Bharat and M. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. *SIGIR-1998*.
8. P. Boldi, M. Santini, and S. Vigna. PageRank as a function of the damping factor. Pages 557–566, *WWW-2005*.
9. A. Borodin, J. S. Rosenthal, G. O. Roberts, and P. Tsaparas, Finding authorities and hubs from link structures on the World Wide Web. *WWW-2001*.
10. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the Web. *WWW-2000*.
11. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30: 107–117, 1998.
12. D. Cai, X. He, J-R. Wen, and W-Y. Ma: Block-level link analysis. *SIGIR-2004*.
13. S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. *WWW-1998*.
14. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *WWW-1999*.

15. Y.-Y. Chen, Q. Gan, and T. Suel. Local methods for estimating PageRank values. *CIKM-2004*.
16. J. Cho and S. Roy. Impact of web search engines on page popularity. *WWW-2004*.
17. J. Cho, S. Roy, and R. Adams. Page quality: In search of an unbiased Web ranking. *SIGMOD-2005*.
18. B. D. Davison. Toward a unification of text and link analysis. Poster abstract of *SIGIR-2003*.
19. P. Diaconis. *Group Representation in Probability and Statistics*. IMS Lecture Series 11, IMS, Hayward, CA, 1988.
20. M. Diligenti, M. Gori, and M. Maggini, Web page scoring systems for horizontal and vertical search. *WWW-2002*.
21. S. Dill, R. Kumar, K. S. McCurley, S. Rajagopalan, D. Sivakumar, and A. Tomkins. Self-similarity in the Web. *VLDB-2001*.
22. R. Fagin, R. Kumar, K. S. McCurley, J. Novak, D. Sivakumar, J. Tomlin, and D. Williamson. Searching the workplace Web. *WWW-2003*.
23. G. Flake, S. Lawrence, and C. L. Giles. Efficient identification of Web communities, Pages 150–160, *KDD-2000.*
24. C. L. Giles. CiteSeer: past, present, and future. *AWIC-2004*.
25. T. Haveliwala. Extrapolation methods for accelerating PageRank computations, *WWW-2003*.
26. R. Jin and S. T. Dumais. Probabilistic combination of content and links. *SIGIR-2001*.
27. S. D. Kamar, T. Haveliwala, C. D. Manning, and G. H. Golub, Extrapolation methods for accelerating PageRank computations. *WWW-2003*.
28. J. Kleinberg. Authoritative sources in a hyperlinked environment. *ACM-SIAM Symposium on Discrete Algorithms*, 1998.
29. J. Kleinberg, S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web as a graph: Measurements, models, and methods. *International Conference on Combinatorics and Computing*, 1999.
30. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Social networks: From the Web to knowledge management. *Web Intelligence*, Pages 367–379, January 2003.
31. S. Lawrence, K. Bollacker, and C. L. Giles. Indexing and retrieval of scientific literature. *CIKM-1999*.
32. R. Lempel and S. Moran, The stochastic approach for link-structure analysis (SALSA) and the TKC effect, *WWW-2000*.
33. F. McSherry. A uniform approach to accelerated PageRank computation. *WWW-2005.*
34. Z. Nie Y. Zhang, J-R. Wen, and W-Y Ma. Object level ranking: Bringing order to Web objects. *WWW-2005*.
35. A. Ntoulas, J. Cho, and C. Olston. What's new on the Web? the evolution of the Web from a search engine perspective. *WWW-2004*.
36. S. Pandey, S. Roy, C. Olston, J. Cho, and S. Chakrabarti. Shuffling a stacked deck: The case for partially randomized ranking of search engine results. *VLDB-2005*.
37. W. Steward. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, 1994.
38. J. Tomlin. A new paradigm for ranking pages on the World Wide Web. *WWW-2003.*
39. P. S. Yu, X. Li, and B. Liu. Adding the temporal dimension to search—a case study in publication search, *WI-2005*.

Chapter 8
Proximity Tracking on Dynamic Bipartite Graphs: Problem Definitions and Fast Solutions

Hanghang Tong, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos

Abstract Large bipartite graphs which evolve and grow over time (e.g., new links arrive, old links die out, or link weights change) arise in many settings, such as social networks, co-citations, market-basket analysis, and collaborative filtering.

Our goal is to monitor (i) the centrality of an individual node (e.g., *who are the most important authors*?) and (ii) the proximity of two nodes or sets of nodes (e.g., *who are the most important authors with respect to a particular conference*?). Moreover, we want to do this efficiently and incrementally and to provide "anytime" answers. In this chapter we propose **pTrack**, which is based on random walks with restart, together with some important modifications to adapt these measures to a dynamic, evolving setting. Additionally, we develop techniques for fast, incremental updates of these measures that allow us to track them continuously, as link updates arrive. In addition, we discuss variants of our method that can handle batch updates, as well as place more emphasis on recent links. Based on proximity tracking, we further proposed **cTrack**, which enables us to track the centrality of the nodes over time. We demonstrate the effectiveness and efficiency of our methods on several real data sets.

8.1 Introduction

Measuring proximity (a.k.a relevance) between nodes on bipartite graphs (see [18] for the formal definition of bipartite graph) is a very important aspect in graph mining and has many real applications, such as ranking, spotting anomaly nodes, connection subgraphs, pattern matching (see Section 8.2 for a detailed review).

Despite their success, most existing methods are designed for static graphs. In many real settings, the graphs are evolving and growing over time, e.g., new links arrive or link weights change. For example, in a user–movie bipartite graph, where the links represent movie ratings given by users, the ratings are usually associated with time information, i.e., the date a user rated the corresponding movie. Similarly,

H. Tong (✉)
Carnegie Mellon University, Pittsburgh PA, 15213, USA
e-mail: htong@cs.cmu.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_8,

in an author–conference bipartite graph, where the links are the number of papers published by the corresponding author in the conference, the papers also have time information, i.e., the year when the paper was published. How should we measure the proximity in such a dynamic setting? What additional benefits can we gain by incorporating time information in proximity measurements?

Here, we address such challenges in multiple dimensions, by focusing on the following questions:

Q1: How to define a good proximity score in a dynamic setting?
Q2: How to incrementally track the proximity scores between nodes of interest as edges are updated?
Q3: What data mining observations do our methods enable?

The answers to these questions are our main contributions:

1: Definitions of proximity and centrality for time-evolving graphs.
2: Two fast update algorithms (*Fast-Single-Update* and *Fast-Batch-Update*), without any quality loss.
3: Two algorithms to incrementally track centrality (*Track-Centrality*) and proximity (*Track-Proximity*) in anytime fashion.
4: Extensive experimental case studies on several real data sets, showing how different queries can be answered, achieving up to **15∼176x** speedup.

The rest of this chapter is organized as follows: we review the related work in Section 8.2. We begin in Section 8.3 with the problem definition and in Section 8.4, we propose our proximity definition for dynamic bipartite graphs. Then, in Section 8.5, we study computational issues thoroughly and propose two fast algorithms, which are the core of computing our dynamic proximity and centrality measurements. The complete algorithms to track proximity (*Track-Proximity*) and centrality (*Track-Centrality*) are presented in Section 8.6. In Section 8.7, we present the experimental valuations on real data sets. Finally, we conclude this chapter in Section 8.8.

8.2 Related Work

In this section, we review the related work, which can be categorized into two parts: static graph mining and dynamic graph mining.

8.2.1 Static Graph Mining

There is a lot of research work on static graph mining, including pattern and law mining [2, 5, 7, 9, 22], frequent substructure discovery [33], influence propagation [16], and community mining [10, 12, 13].

In terms of centrality, Google's PageRank algorithm [23] is the most related. The proposed *Track-Centrality* can actually be viewed as its generalization for dynamic bipartite graphs. As for proximity, the closest work is random walk with restart [15, 24, 32]. The proposed *Track-Proximity* is its generalization for dynamic bipartite graphs. Other representative proximity measurements on static graphs include the sink-augmented delivered current [8], cycle-free effective conductance [17], survivable network [14], and direction-aware proximity [31]. Although we focus on random walk with restart in this chapter, our fast algorithms can be easily adapted to other random walk based measurements, such as [8, 31]. Also, there are a lot of applications of proximity measurements. Representative work includes connection subgraphs [8, 17, 29], neighborhood formation in bipartite graphs [27], content-based image retrieval [15], cross-modal correlation discovery [24], the BANKS system [1], link prediction [20], pattern matching [30], detecting anomalous nodes and links in a graph [27], ObjectRank [4], and RelationalRank [11].

8.2.2 Dynamic Graph Mining

More recently, there is an increasing interest in mining time-evolving graphs, such as densification laws and shrinking diameters [19], community evolution [3], dynamic tensor analysis [28], and dynamic communities [6, 26]. To the best of our knowledge, there is no previous work on proximity for time-evolving graphs. Remotely related work in the sparse literature on the topic is [21]. However, we have a different setting and focus compared with [21]: we aim to incrementally track the proximity and centrality for nodes of interest by quickly updating the core matrix (as well as the adjacency matrices), while in [21] the authors focus on efficiently using time information by adding time as explicit nodes in the graph.

8.3 Problem Definitions

Table 8.1 lists the main symbols we use throughout the chapter. Following standard notation, we use capital letters for matrices $\mathbf{M}$ and arrows for vectors. We denote the transpose with a prime (i.e., $\mathbf{M}'$ is the transpose of $\mathbf{M}$), and we use parenthesized superscripts to denote time (e.g., $\mathbf{M}^{(t)}$ is the time-aggregate adjacency matrix at time t). When we refer to a static graph or, when time is clear from the context, we omit the superscript (t). We use subscripts to denote the size of matrices/vectors (e.g. $\mathbf{0}_{n\times l}$ means a matrix of size $n \times l$, whose elements are all zero). Also, we represent the elements in a matrix using a convention similar to Matlab, e.g., $\mathbf{M}(i, j)$ is the element at the ith row and jth column of the matrix $\mathbf{M}$, and $\mathbf{M}(i, :)$ is the ith row of $\mathbf{M}$ (i.e., $\mathbf{M}(i, :)$ contains all the edges from the ith type 1 object to all the type 2 objects.) Without loss of generality, we assume that the numbers of type 1 and type

Table 8.1 Symbols

Symbol	Definition and description
$\mathbf{M}^{(t)}$	$n \times l$ time-aggregate adjacency matrix at time t
$\mathbf{S}^{(t)}$	$n \times l$ slice matrix at time t
$\Delta\mathbf{M}^{(t)}$	$n \times l$ difference matrix at time t
$\mathbf{D}_1^{(t)}$	$n \times n$ out-degree matrix for type 1 object, i.e. $\mathbf{D}_1^{(t)}(i,i) = \sum_{j=1}^{n} \mathbf{M}^{(t)}(i,j)$, and $\mathbf{D}_1^{(t)}(i,j) = 0\ (i \neq j)$
$\mathbf{D}_2^{(t)}$	$l \times l$ out-degree matrix for type 2 object, *i.e.*$\mathbf{D}_2^{(t)}(i,i) = \sum_{j=1}^{n} \mathbf{M}^{(t)}(j,i)$, and $\mathbf{D}_2^{(t)}(i,j) = 0\ (i \neq j)$
$\mathbf{I}$	identity matrix
$\mathbf{0}$	a matrix with all elements equal to 0
$\mathbf{1}$	a matrix with all elements equal to 1
n, l	number of nodes for type 1 and type 2 objects, respectively ($n > l$)
m	number of edges in the bipartite graph
c	$(1-c)$ is fly-out probability for random walk with restart (set to be 0.95 in the paper)
$r_{i,j}^{(t)}$	proximity from node i to node j at time t

2 objects are fixed (i.e., n and l are constant for all time steps); if not, we can reserve rows/columns with zero elements as necessary.

At each time step, we observe a set of new edges or edge weight updates. These represent the link information that is available at the finest time granularity. We use the *time-slice matrix*, or *slice matrix* for brevity, $\mathbf{S}^{(t)}$ to denote the new edges and additional weights that appear at time step t. For example, given a set of authors and annual conferences, the number of papers that author i publishes in conference j during year t is the entry $\mathbf{S}^{(t)}(i,j)$. In this chapter, we focus only on the case of edge additions and weight increases (e.g., authors always publish new papers, and users always rate more movies). However, the ideas we develop can be easily generalized to handle other types of link updates, such as links deletions or edge weights decreases.

Given the above notion, a dynamic, evolving graph can be naturally defined as a sequence of observed new edges and weights, $\mathbf{S}^{(1)}, \mathbf{S}^{(2)}, \ldots, \mathbf{S}^{(t)}, \ldots$. However, the information for a single time slice may be too sparse for meaningful analysis, and/or users typically want to analyze larger portions of the data to observe interesting patterns and trends. Thus, from a sequence of slice matrices observed so far, $\mathbf{S}^{(j)}$ for $1 \leq j \leq t$, we construct a bipartite graph by aggregating time slices. We propose three different aggregation strategies, which place different emphasis on edges based on their age. In all cases, we use the term *time-aggregate adjacency matrix* (or *adjacency matrix* for short), denoted by $\mathbf{M}^{(t)}$, for the adjacency matrix of the bipartite graph at time step t. We will introduce the aggregation strategies in the next section).

Finally, to simplify the description of our algorithms, we introduce the *difference matrix* $\Delta\mathbf{M}^{(t)}$, which is the difference between two consecutive adjacency matrices, i.e., $\Delta\mathbf{M}^{(t)} \triangleq \mathbf{M}^{(t)} - \mathbf{M}^{(t-1)}$. Note that, depending on the aggregation strategy, difference matrix $\Delta\mathbf{M}^{(t)}$ may or may not be equal to the slice matrix $\mathbf{S}^{(t)}$.

An important observation from many real applications is that despite the large size of the graphs involved (with hundreds of thousands or millions of nodes and edges), the intrinsic dimension (or, effective rank) of their corresponding adjacency matrices is usually relatively small, primarily because there are relatively fewer objects of one type. For example, on the author–conference graph from the **AC** data set, although we have more than 400,000 authors and about 2 million edges, there are only $\sim$ 3500 conferences. In the user–movie graph from the **NetFlix** data set, although we have about 2.7 million users with more than 100 million edges, there are only 17,700 movies. We use the term *skewed* to refer to such bipartite graphs, i.e., $n, m \gg l$.

With the above notation, our problems (**pTrack** and **cTrack**) can be formally defined as follows:

Problem 1 pTrack

Given: (i) a large, skewed time-evolving bipartite graph $\{\mathbf{S}^{(t)}, t = 1, 2, ...\}$, and (ii) the query nodes of interest $(i, j, ...)$
Track: (i) the top-k most related objects for each query node at each time step and (ii) the proximity score (or the proximity rank) for any two query nodes at each time step.

There are two different kinds of tracking tasks in **pTrack**, both of which are related to proximity. For example, in a time-evolving author–conference graph we can track "*What are the major conferences for John Smith in the past 5 years?*" which is an example of task (i); or "*How much credit (importance) has John Smith accumulated in the KDD Conference so far?*" which is an example of task (ii). We will propose an algorithm (*Track-Proximity*) in Section 8.6 to deal with **pTrack**.

Problem 2 cTrack

Given: (i) a large, skewed time-evolving bipartite graph $\{\mathbf{S}^{(t)}, t = 1, 2, ...\}$ and (ii) the query nodes of interest $(i, j, ...)$
Track: (i) the top-k most central objects in the graph, for each query node and at each time step and (ii) the centrality (or the rank of centrality), for each query node at each time step.

In **cTrack**, there are also two different kinds of tracking tasks, both of which are related to centrality. For example, in the same time-evolving author–conference graph, we can track "*How influential is author-A over the years?*" which corresponds to task (i) or "*Who are the top-10 influential authors over the years?*" which corresponds to task (ii). Note that in task (ii) of **cTrack**, we do not need the query nodes as inputs. We will propose another algorithm (*Track-Centrality*) in Section 8.1.6 to deal with **cTrack**.

For all these tasks (**pTrack** and **cTrack**), we want to provide anytime answers. That is, we want to quickly maintain up-to-date answers as soon as we observe a new slice matrix $\mathbf{S}^{(t)}$.

8.4 Dynamic Proximity and Centrality: Definitions

In this section, we introduce our proximity and centrality definitions for dynamic bipartite graphs. We begin by reviewing random walk with restart, which is a good proximity measurement for static graphs. We then extend it to the dynamic setting by (1) using different ways to aggregate edges from different time steps, that is to place different emphasis on more recent links and (2) using *degree-preservation* to achieve monotonicity for dynamic proximity.

8.4.1 Background: Static Setting

Among many others, one very successful method to measure proximity is random walk with restart (RWR), which has been receiving increasing interest in recent years—see Section 8.2 for a detailed review.

For a static bipartite graph, random walk with restart is defined as follows: Consider a random particle that starts from node i. The particle iteratively transits to its neighbors with probability proportional to the corresponding edge weights. Also at each step, the particle returns to node i with some restart probability $(1 - c)$. The proximity score from node i to node j is defined as the steady-state probability $r_{i,j}$ that the particle will be on node j [24]. Intuitively, $r_{i,j}$ is the fraction of time that the particle starting from node i will spend on each node j of the graph, after an infinite number of steps.

If we represent the bipartite graph as a unipartite graph with the following square adjacency matrix $\mathbf{W}$ and degree matrix $\mathbf{D}$:

$$\mathbf{W} = \begin{pmatrix} \mathbf{0}_{n\times n} & \mathbf{M} \\ \mathbf{M}' & \mathbf{0}_{l\times l} \end{pmatrix}$$
$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 & \mathbf{0}_{n\times l} \\ \mathbf{0}_{l\times n} & \mathbf{D}_2 \end{pmatrix}, \tag{8.1}$$

then, all the proximity scores $r_{i,j}$ between all possible node pairs i, j are determined by the matrix $\mathbf{Q}$:

$$r_{i,j} = \mathbf{Q}(i, j).$$
$$\mathbf{Q} = (1 - c) \cdot \left(\mathbf{I}_{(n+l)\times(n+l)} - c\mathbf{D}^{-1}\mathbf{W}\right)^{-1} \tag{8.2}$$

Based on the dynamic proximity as in (8.2), we define the centrality for a given source node s as the average proximity score from all nodes in the graph (including s itself) to s. For simplicity, we ignore the time step superscript. That is,

$$\text{centrality}(s) \triangleq \frac{\sum_{i=1}^{n+l} r_{i,s}}{n+l}. \tag{8.3}$$

8.4.2 Dynamic Proximity

Since centrality is defined in terms of proximity, we will henceforth focus only on the latter. In order to apply the random walk with restart (see (8.2)) to the dynamic setting, we need to address two subtle but important points.

The first is how to update the adjacency matrix $\mathbf{M}^{(t)}$ based on the observed slice matrix $\mathbf{S}^{(t)}$. As mentioned before, usually it is not enough to consider only the current slice matrix $\mathbf{S}^{(t)}$. For example, examining publications from conferences in a single year may lead to proximity scores that vary widely and reflect more "transient" effects (such as a bad year for an author), rather than "true" shifts in his affinity to research areas (for example, a shift of interest from databases to data mining or a change of institutions and collaborators). Similarly, examining movie ratings from a single day may not be sufficient to accurately capture the proximity of, say, two users in terms of their tastes. Thus, in Section 8.3.2.1, we propose three different strategies to aggregate slices into an adjacency matrix $\mathbf{M}^{(t)}$ or, equivalently, to update $\mathbf{M}^{(t)}$. Note, however, that single-slice analysis can be viewed as a special case of the "sliding window" aggregation strategy.

The second point is related to the "monotonicity" of proximity versus time. In a dynamic setting with only link additions and weight increases (i.e., $\mathbf{S}^{(t)}(i, j) \geq 0$, for all time steps t and nodes i, j), in many applications it is desirable that the proximity between any two nodes does not drop. For example, consider an author–conference bipartite graph, where edge weights represent the number of papers that an author has published in the corresponding conference. We would like a proximity measure that represents the total contribution/credit that an author has accumulated in each conference. Intuitively, this score should not decrease over time.

8.4.2.1 Updating the Adjacency Matrix

As explained above, it is usually desirable to analyze multiple slices together, placing different emphasis on links based on their age. For completeness, we describe three possible aggregation schemes.

Global Aggregation. The first way to obtain the adjacency matrix $\mathbf{M}^{(t)}$ is to simply add the new edges or edge weights in $\mathbf{S}^{(t)}$ to the previous adjacency matrix $\mathbf{M}^{(t-1)}$ as follows:

$$\mathbf{M}^{(t)} = \sum_{j=1}^{t} \mathbf{S}^{(j)}.$$

We call this scheme *global aggregation*. It places equal emphasis on all edges from the beginning of time and, only in this case, $\Delta\mathbf{M}^{(t)} = \mathbf{S}^{(t)}$. Next, we define schemes that place more emphasis on recent links. For both of these schemes, $\Delta\mathbf{M}^{(t)} \neq \mathbf{S}^{(t)}$.

Sliding Window. In this case, we only consider the edges and weights that arrive in the past *len* time steps, where the parameter *len* is the length of the sliding window:

$$\mathbf{M}^{(t)} = \sum_{j=\max\{1,\ t-len+1\}}^{t} \mathbf{S}^{(j)}$$

Exponential Weighting. In this case, we "amplify" the new edges and weights at time t by an exponential factor $\beta^j (\beta > 1)$: $\mathbf{M}^{(t)} = \sum_{j=1}^{t} \beta^j \mathbf{S}^{(j)}$.

8.4.2.2 Fixed Degree Matrix

In a dynamic setting, if we apply the actual degree matrix $\mathbf{D}^{(t)}$ to (8.2) at time t, the monotonicity property will not hold. To address this issue, we propose to use degree-preservation [17, 31]. That is, we use the same degree matrix $\tilde{\mathbf{D}}$ at all time steps.

Thus, our proximity $r_{i,j}^{(t)}$ from node i to node j at time step t is formally defined as in (8.4). The adjacency matrix $\mathbf{M}^{(t)}$ is computed by any update method in the above section and the fixed degree matrix $\tilde{\mathbf{D}}$ is set to be a constant (a) times the degree matrix at the first time step—we always set $a = 1000$ in this chapter.

$$\begin{aligned} r_{i,j}^{(t)} &= \mathbf{Q}^{(t)}(i, j) \\ \mathbf{Q}^{(t)} &= (1-c) \cdot \left(\mathbf{I}_{(n+l)\times(n+l)} - c\tilde{\mathbf{D}}^{-1}\mathbf{W}^{(t)}\right)^{-1}. \\ \mathbf{W}^{(t)} &= \begin{pmatrix} \mathbf{0}_{n\times n} & \mathbf{M}^{(t)} \\ \mathbf{M}'^{(t)} & \mathbf{0}_{l\times l} \end{pmatrix} \\ \tilde{\mathbf{D}} &= a \cdot \mathbf{D}^{(1)} \end{aligned} \tag{8.4}$$

We have the following lemma for our dynamic proximity (8.4). By the lemma 1, if the actual degree $\mathbf{D}^{(t)}(i, i)$ does not exceed the fixed degree $\tilde{\mathbf{D}}(i, i)$ (condition 2), then the proximity between any two nodes will never drop as long as the edge weights in adjacency matrix $\mathbf{M}^{(t)}$ do not drop (condition 1).

Lemma 1 *Monotonicity Property of Dynamic Proximity If* (1) *all elements in the difference matrix* $\Delta\mathbf{M}^{(t)}$ *are non-negative and* (2) $\mathbf{D}^{(t)}(i, i) \leq \tilde{\mathbf{D}}(i, i)$ $(i = 1, 2, ..., (n + l))$*, then we have* $r_{i,j}^{(t)} \geq r_{i,j}^{(t-1)}$ *for any two nodes* (i, j).

Proof First of all, since $\mathbf{D}^{(t)}(i, i) \leq \tilde{\mathbf{D}}(i, i)$, we have $\|c\tilde{\mathbf{D}}^{-1}\mathbf{W}^{(t)}\|^k \to 0$ as $k \to \infty$. Therefore, we have $\mathbf{Q}^{(t)} = (1-c)\sum_{k=0}^{\infty}\left(c\tilde{\mathbf{D}}^{-1}\mathbf{W}^{(t)}\right)^k$. On the other hand, since all elements in the difference matrix $\Delta\mathbf{M}^{(t)}$ are non-negative, we have $\mathbf{W}^{(t)}(i, j) \geq \mathbf{W}^{(t-1)}(i, j)$ for any two nodes (i, j). Therefore, we have $\mathbf{Q}^{(t)}(i, j) \geq \mathbf{Q}^{(t-1)}(i, j)$ for any two nodes (i, j), which completes the proof. □

Finally, we should point out that a, $\mathbf{D}$ and the non-negativity of $\mathbf{M}$ are relevant only if a monotonic score is desired. Even without these assumptions, the correctness or efficiency of our proposed algorithms is not affected. If non-monotonic scores are permissible, none of these assumptions are necessary. And also, the lemma only applies when there is no edge deletion (since we require that the difference matrix $\Delta\mathbf{M}^{(t)}$ are non-negative).

8.5 Dynamic Proximity: Computations

8.5.1 Preliminaries: BB_LIN on Static Graphs

In this section, we introduce our fast solutions to efficiently track dynamic proximity. We will start with BB_LIN [32], a fast algorithm for static, skewed bipartite graphs. We then extend it to the dynamic setting.

One problem with random walk with restart is computational efficiency, especially for large graphs. According to the definition (8.4), we need to invert an $(n + l) \times (n + l)$ matrix. This operation is prohibitively slow for large graphs. In [32], the authors show that for skewed, static bipartite graphs, we only need to pre-compute and store a matrix inversion of size $l \times l$ to get all possible proximity scores (see [32] for the proof). BB_LIN, which is the starting point for our fast algorithms, is summarized in Algorithm 1.

Algorithm 1 BB_LIN

Input: The adjacency matrix at time t, as in equation (8.1); and the query nodes i and j.
Output: The proximity $r_{i,j}$ from node i to node j.

1: **Pre-Computation Stage(Off-Line):**
2: normalize for type 1 objects: $\mathbf{Mr} = \mathbf{D}_1^{-1} \cdot \mathbf{M}$
3: normalize for type 2 objects: $\mathbf{Mc} = \mathbf{D}_2^{-1} \cdot \mathbf{M}'$
4: compute the core matrix: $\mathbf{C} = (\mathbf{I} - c^2\mathbf{Mc} \cdot \mathbf{Mr})^{-1}$
5: store the matrices: **Mr**, **Mc**, and **C**.
6: **Query Stage (On-Line):**
7: **Return:** $r_{i,j} =$ **GetQij**($\mathbf{C}$, **Mr**, **Mc**, i, j, c)

Based on Algorithm 1, we only need to pre-compute and store a matrix inversion $\mathbf{C}$ of size $l \times l$. For skewed bipartite graphs ($l \ll m, n$), $\mathbf{C}$ is much cheaper to pre-compute and store. For example, on the entire **NetFlix** user–movie bipartite graph, which contains about 2.7 M users, about 18 K movies and more than 100 M edges (see Section 8.6 for the detailed description of the data set), it takes 1.5 h to pre-compute the 18 K $\times$ 18 K matrix inversion $\mathbf{C}$. For pre-computation stage, this is quite acceptable.

Algorithm 2 GetQij

Input: The core matrix **C**, the normalized adjacency matrices **Mr** (for type 1 objects), and **Mc** (for type 2), and the query nodes i and j ($1 \leq i, j \leq (n+l)$).
Output: The proximity $r_{i,j}$ from node i to node j

1: **if** $i \leq n$ and $j \leq n$ **then**
2: $q(i, j) = 1(i = j) + c^2\mathbf{Mr}(i, :) \cdot \mathbf{C} \cdot \mathbf{Mc}(:, j)$
3: **else if** $i \leq n$ and $j > n$ **then**
4: $q(i, j) = c\mathbf{Mr}(i, :) \cdot \mathbf{C}(:, j - n)$
5: **else if** $i > n$ and $j \leq n$ **then**
6: $q(i, j) = c\mathbf{C}(i - n, :) \cdot \mathbf{Mc}(:, j)$
7: **else**
8: $q(i, j) = \mathbf{C}(i - n, j - n)$
9: **end if**
10: **Return:** $r_{i,j} = (1 - c)q(i, j)$

On the other hand, in the online query stage, we can get any proximity scores using the function **GetQij**.[1] This stage is also cheap in terms of computation. For example, to output a proximity score between two type 1 objects (step 2 in **GetQij**) , only one sparse vector–matrix multiplication and one vector–vector multiplication are needed. For a proximity score between one type 1 object and one type 2 object, only one sparse vector–vector multiplication (steps 4 and 6) is necessary. Finally, for a proximity score between two type 2 objects (step 8), only retrieving one element in the matrix **C** is needed. As an example, on the **NetFlix** data set, it takes less than 1 s to get one proximity score. Note that all possible proximity scores are determined by the matrix **C** (together with the normalized adjacency matrices **Mr** and **Mc**). We thus refer to the matrix **C** as the the *core matrix*.

8.5.2 Challenges for Dynamic Setting

In a dynamic setting, since the adjacency matrix changes over time, the core matrix $\mathbf{C}^{(t)}$ is no longer constant. In other words, the steps 1–4 in Algorithm 1 themselves become a part of the online stage since we need to update the core matrix $\mathbf{C}^{(t)}$ at each time step. If we still rely on the straightforward strategy (i.e., the steps 1–4 in Algorithm 1) to update the core matrix (referred to as "Straight-Update"), the total computational complexity for each time step is $O(l^3 + m \cdot l)$. Such complexity is undesirable for the online stage. For example, 1.5 h to recompute the core matrix for the **NetFlix** data set is unacceptably long.

Thus, our goal is to efficiently update the core matrix $\mathbf{C}^{(t)}$ at time step t, based on the previous core matrix $\mathbf{C}^{(t-1)}$ and the difference matrix $\Delta\mathbf{M}^{(t)}$. For simplicity, we shall henceforth assume the use of the global aggregation scheme to update the

[1] Note that in step 2 of **GetQij**, 1(.) is the indicator function, i.e. it is 1 if the condition in (.) is true and 0 otherwise.

adjacency matrix. However, the ideas can be easily applied to the other schemes, sliding window and exponential weighting.

8.5.3 Our Solution 1: Single Update

Next, we describe a fast algorithm (*Fast-Single-Update*) to update the core matrix $\mathbf{C}^{(t)}$ at time step t, if only one edge (i_0, j_0) changes at time t. In other words, there is only one non-zero element in $\Delta\mathbf{M}^{(t)}$: $\Delta\mathbf{M}^{(t)}(i_0, j_0) = w_0$. To simplify the description of our algorithm, we present the difference matrix $\Delta\mathbf{M}^{(t)}$ as a from-to list: $[i_0, j_0, w_0]$.

The correctness of *Fast-Single-Update* is guaranteed by the following theorem:

Theorem 1 *Correctness of Fast-Single-Update The matrix* $\mathbf{C}^{(t)}$ *maintained by* Fast-Single-Update *is exactly the core matrix at time step t, i.e.,* $\mathbf{C}^{(t)} = \left(\mathbf{I} - c^2\mathbf{Mc}^{(t)}\mathbf{Mr}^{(t)}\right)^{-1}$.

Proof First of all, since only one edge (i_0, j_0) is updated at time t, only the i_0th row of the matrix $\mathbf{Mr}^{(t)}$ and the i_0th column of the matrix $\mathbf{Mc}^{(t)}$ change at time t.

Let $\mathbf{V}^{(t)} = c^2\mathbf{Mc}^{(t)} \cdot \mathbf{Mr}^{(t)}$ and $\mathbf{V}^{(t-1)} = c^2\mathbf{Mc}^{(t-1)} \cdot \mathbf{Mr}^{(t-1)}$. By the spectral representation of $\mathbf{V}^{(t)}$ and $\mathbf{V}^{(t-1)}$, we have the following equation:

$$\begin{aligned}\mathbf{V}^t &= c^2 \sum_{k=1}^{n} \mathbf{Mc}^{(t)}(:, k) \cdot \mathbf{Mr}^{(t)}(k, :), \\ &= \mathbf{V}^{t-1} + \delta \end{aligned} \tag{8.5}$$

where δ indicates the difference between $\mathbf{V}^{(t)}$ and $\mathbf{V}^{(t-1)}$. This gives us

$$\delta = \sum_{s=0}^{1} (-1)^s \cdot c^2\mathbf{Mc}^{(t)}(:, i_0) \cdot \mathbf{Mr}^{(t-s)}(i_0, :) = \mathbf{X} \cdot \mathbf{Y},$$

where the matrices $\mathbf{X}$ and $\mathbf{Y}$ are defined in steps 4–6 of Algorithm 3. Putting all the above together, we have

$$\mathbf{C}^t = (\mathbf{I} - \mathbf{V}^t)^{-1} = (\mathbf{I} - \mathbf{V}^{t-1} - \mathbf{X} \cdot \mathbf{Y})^{-1}. \tag{8.6}$$

Applying the Sherman–Morrison lemma [25] to (8.6), we have

$$\mathbf{C}^{(t)} = \mathbf{C}^{(t-1)} + \mathbf{C}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{C}^{(t-1)},$$

where the 2×2 matrix $\mathbf{L}$ is defined in step 7 of Algorithm 3. This completes the proof. □

Algorithm 3 *Fast-Single-Update*

Input: The core matrix $\mathbf{C}^{(t-1)}$, the normalized adjacency matrices $\mathbf{Mr}^{(t-1)}$ (for type 1 objects) and $\mathbf{Mc}^{(t-1)}$ (for type 2 objects) at time step $t-1$, and the difference list $[i_0, j_0, w_0]$ at the time step t.

Output: The core matrix $\mathbf{C}^{(t)}$, the normalized adjacency matrices $\mathbf{Mr}^{(t)}$ and $\mathbf{Mc}^{(t)}$ at time step t.

1: $\mathbf{Mr}^{(t)} = \mathbf{Mr}^{(t-1)}$, and $\mathbf{Mc}^{(t)} = \mathbf{Mc}^{(t-1)}$.
2: $\mathbf{Mr}^{(t)}(i_0, j_0) = \mathbf{Mr}^{(t)}(i_0, j_0) + \frac{w_0}{\tilde{\mathbf{D}}(i_0, i_0)}$
3: $\mathbf{Mc}^{(t)}(j_0, i_0) = \mathbf{Mc}^{(t)}(j_0, i_0) + \frac{w_0}{\tilde{\mathbf{D}}(j_0+n, j_0+n)}$
4: $\mathbf{X} = \mathbf{0}_{l\times 2}$, and $\mathbf{Y} = \mathbf{0}_{2\times l}$
5: $\mathbf{X}(:, 1) = \mathbf{Mc}^{(t)}(:, i_0)$, and $\mathbf{X}(j_0, 2) = \frac{w_0}{\tilde{\mathbf{D}}(j_0+n, j_0+n)}$
6: $\mathbf{Y}(1, j_0) = \frac{c^2 \cdot w_0}{\tilde{\mathbf{D}}(i_0, i_0)}$, and $\mathbf{Y}(2, :) = c^2 \cdot \mathbf{Mr}^{(t-1)}(i_0, :)$
7: $\mathbf{L} = (\mathbf{I}_{2\times 2} - \mathbf{Y} \cdot \mathbf{C}^{(t-1)} \cdot \mathbf{X})^{-1}$
8: $\mathbf{C}^{(t)} = \mathbf{C}^{(t-1)} + \mathbf{C}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{C}^{(t-1)}$

Fast-Single-Update is significantly more computationally efficient, as shown by the next lemma. In particular, the complexity of *Fast-Single-Update* is only $O(l^2)$, as opposed to $O(l^3 + ml)$ for the straightforward method.

Lemma 2 *Efficiency of Fast-Single-Update The computational complexity of Fast-Single-Update is* $O(l^2)$.

Proof The computational cost for step 1 is $O(l^2)$. It is $O(1)$ for steps 2 and 3, $O(l)$ for steps 4–6 and $O(l^2)$ for steps 7 and 8. Putting it together, we have that the total cost for *Fast-Single-Update* is $O(l^2)$, which completes the proof. □

8.5.4 Our Solutions 2: Batch Update

In many real applications, more than one edges typically change at each time step. In other words, there are multiple non-zero elements in the difference matrix $\Delta\mathbf{M}^{(t)}$. Suppose we have a total of $\hat{m}$ edge changes at time step t. An obvious choice is to repeatedly call *Fast-Single-Update* $\hat{m}$ times.

An important observation from many real applications is that it is unlikely these $\hat{m}$ edges are randomly distributed. Instead, they typically form a low-rank structure. That is, if these $\hat{m}$ edges involve $\hat{n}$ type 1 objects and $\hat{l}$ type 2 objects, we have $\hat{n} \ll \hat{m}$ or $\hat{l} \ll \hat{m}$. For example, in an author–conference bipartite graph, we will often add a group of $\hat{m}$ new records into the database at one time step. In most cases, these new records only involve a small number of authors and/or conferences—see Section 8.6 for the details. In this section, we show that we can do a single batch update (*Fast-Batch-Update*) on the core matrix. This is much more efficient than either doing $\hat{m}$ single updates repeatedly or recomputing the core matrix from scratch. The main advantage of our approach lies on the observation that the difference matrix has low rank, and our upcoming algorithm needs time proportional to the *rank*, as opposed to the number of changed edges $\hat{m}$. This holds in real settings,

because when a node is modified, several of its edges are changed (e.g., an author publishes several papers in a given conference each year).

Let $\mathcal{I} = \{i_1, ..., i_{\hat{n}}\}$ be the indices of the involved type 1 objects. Similarly, let $\mathcal{J} = \{j_1, ..., j_{\hat{l}}\}$ be the indices of the involved type 2 objects. We can represent the difference matrix $\Delta\mathbf{M}^{(t)}$ as an $\hat{n} \times \hat{l}$ matrix. In order to simplify the description of the algorithm, we define two matrices $\Delta\mathbf{M}r$ and $\Delta\mathbf{M}c$ as follows:

$$\begin{aligned} \Delta\mathbf{M}r(k, s) &= \frac{\Delta\mathbf{M}^{(t)}(i_k, j_s)}{\tilde{\mathbf{D}}(i_k, i_k)} \\ \Delta\mathbf{M}c(s, k) &= \frac{\Delta\mathbf{M}^{(t)}(j_s, i_k)}{\tilde{\mathbf{D}}(j_s + n, j_s + n)} \\ (k = 1, ..., \hat{n}, s &= 1, ..., \hat{l}). \end{aligned} \tag{8.7}$$

The correctness of *Fast-Batch-Update* is guaranteed by the following theorem:

Theorem 2 Delta Matrix Inversion Theorem *The matrix* $\mathbf{C}^{(t)}$ *maintained by* Fast-Batch-Update *is exactly the core matrix at time step t, i.e.,* $\mathbf{C}^{(t)} = \left(\mathbf{I} - c^2\mathbf{M}\mathbf{c}^{(t)}\mathbf{M}\mathbf{r}^{(t)}\right)^{-1}$.

Proof Let $\mathbf{V}^{(t)} = c^2\mathbf{M}\mathbf{c}^{(t)} \cdot \mathbf{M}\mathbf{r}^{(t)}$ and $\mathbf{V}^{(t-1)} = c^2\mathbf{M}\mathbf{c}^{(t-1)} \cdot \mathbf{M}\mathbf{r}^{(t-1)}$. Similar as the proof for Theorem 1, we have

$$\mathbf{V}^{(t)} = \mathbf{V}^{(t-1)} - \mathbf{X} \cdot \mathbf{Y}, \tag{8.8}$$

where the matrices $\mathbf{X}$ and $\mathbf{Y}$ are defined in steps 6–21 of Algorithm 4.

Applying the Sherman–Morrison lemma [25] to (8.8), we have

$$\mathbf{C}^{(t)} = \mathbf{C}^{(t-1)} + \mathbf{C}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{C}^{(t-1)},$$

where the $2\hat{k} \times 2\hat{k}$ matrix $\mathbf{L}$ is defined in step 22 of Algorithm 4. This completes the proof. □

The efficiency of *Fast-Single-Update* is given by the following lemma. Compared to the straightforward recomputation which is $O(l^3 + ml)$, *Fast-Batch-Update* is $O(\min(\hat{\mathrm{l}}, \hat{\mathrm{n}}) \cdot \mathrm{l}^2 + \hat{\mathrm{m}})$. Since $\min(\hat{\mathrm{l}}, \hat{\mathrm{n}}) < \mathrm{l}$ always holds, as long as we have $\hat{m} < m$, *Fast-Single-Update* is always more efficient. On the other hand, if we do $\hat{m}$ repeated single updates using *Fast-Single-Update*, the computational complexity is $O(\hat{m}l^2)$. Thus, since typically $\min(\hat{\mathrm{l}}, \hat{\mathrm{n}}) \ll \hat{m}$, *Fast-Batch-Update* is much more efficient in this case.

Lemma 3 *Efficiency of Fast-Batch-Update The computational complexity of Fast-Batch-Update is* $O(\min(\hat{\mathrm{l}}, \hat{\mathrm{n}}) \cdot \mathrm{l}^2 + \hat{\mathrm{m}})$.

Proof Similar as the proof for lemma 2. Note that the linear term $O(\hat{m})$ comes from (8.7), since we need to scan the non-zero elements of the difference matrix $\Delta\mathbf{M}^{(t)}$.

Algorithm 4 *Fast-Batch-Update*

Input: The core matrix $\mathbf{C}^{(t-1)}$, the normalized adjacency matrices $\mathbf{Mr}^{(t-1)}$ (for type 1 objects) and $\mathbf{Mc}^{(t-1)}$ (for type 2 objects) at time step $t-1$, and the difference matrix $\Delta\mathbf{M}^{(t)}$ at the time step t

Output: The core matrix $\mathbf{C}^{(t)}$, the normalized adjacency matrices $\mathbf{Mr}^{(t)}$ and $\mathbf{Mc}^{(t)}$ at time step t.

1: $\mathbf{Mr}^{(t)} = \mathbf{Mr}^{(t-1)}$, and $\mathbf{Mc}^{(t)} = \mathbf{Mc}^{(t-1)}$.
2: define $\Delta\mathbf{M}r$ and $\Delta\mathbf{M}c$ as in equation (8.7)
3: $\mathbf{Mr}^{(t)}(\mathcal{I}, \mathcal{J}) = \mathbf{Mr}^{(t)}(\mathcal{I}, \mathcal{J}) + \Delta\mathbf{M}r$
4: $\mathbf{Mc}^{(t)}(\mathcal{J}, \mathcal{I}) = \mathbf{Mc}^{(t)}(\mathcal{J}, \mathcal{I}) + \Delta\mathbf{M}c$
5: let $\hat{k} = \min(\hat{l}, \hat{n})$. let $\mathbf{X} = \mathbf{0}_{m\times 2\hat{k}}$, and $\mathbf{Y} = \mathbf{0}_{2\hat{k}\times m}$
6: **if** $\hat{l} < \hat{n}$ **then**
7: $\quad\mathbf{X}(:, 1:\hat{l}) = \mathbf{Mc}^{(t-1)}(:, \mathcal{I}) \cdot \Delta\mathbf{M}r$
8: $\quad\mathbf{Y}(\hat{l}+1:2\hat{l}, :) = \Delta\mathbf{M}c \cdot \mathbf{Mr}^{(t-1)}(\mathcal{I}, :)$
9: $\quad\mathbf{X}(\mathcal{J}, 1:\hat{l}) = \mathbf{X}(\mathcal{J}, 1:\hat{l}) + \Delta\mathbf{M}c \cdot \Delta\mathbf{M}r$
10: $\quad\mathbf{X}(\mathcal{J}, 1:\hat{l}) = \mathbf{X}(\mathcal{J}, 1:\hat{l}) + \mathbf{Y}(\hat{l}+1:2\hat{l}, \mathcal{J})$
11: $\quad\mathbf{Y}(\hat{l}+1:2\hat{l}, \mathcal{J}) = 0$
12: $\quad$**for** $k = 1:\hat{k}$ **do**
13: $\quad\quad$set $\mathbf{Y}(k, j_k) = 1$, and $\mathbf{X}(j_k, k+\hat{k}) = 1$
14: $\quad$**end for**
15: $\quad$set $\mathbf{X} = c^2 \cdot \mathbf{X}$, and $\mathbf{Y} = c^2 \cdot \mathbf{Y}$
16: **else**
17: $\quad\mathbf{X}(:, 1:\hat{n}) = \mathbf{Mc}^{(t)}(:, \mathcal{I})$
18: $\quad\mathbf{X}(\mathcal{J}, \hat{n}+1:2\hat{n}) = \Delta\mathbf{M}c$
19: $\quad\mathbf{Y}(1:\hat{n}, \mathcal{J}) = c^2 \cdot \Delta\mathbf{M}r$
20: $\quad\mathbf{Y}(\hat{n}+1:2\hat{n}, :) = c^2 \cdot \mathbf{Mr}^{(t-1)}(\mathcal{I}, :)$
21: **end if**
22: $\mathbf{L} = (\mathbf{I}_{2\hat{k}\times 2\hat{k}} - \mathbf{Y} \cdot \mathbf{C}^{(t-1)} \cdot \mathbf{X})^{-1}$
23: $\mathbf{C}^{(t)} = \mathbf{C}^{(t-1)} + \mathbf{C}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{C}^{(t-1)}$

And the term of $O(\min(\hat{l}, \hat{n}) \cdot l^2)$ comes from the steps 22 and 23 of *Fast-Batch-Update*. □

8.6 Dynamic Proximity: Applications

In this section, we give the complete algorithms for the two applications we posed in Section 8.2, that is, *Track-Centrality* and *Track-Proximity*. For each case, we can track top-k queries over time. For *Track-Centrality*, we can also track the centrality (or the centrality rank) for an individual node. For *Track-Proximity*, we can also track the proximity (or the proximity rank) for a given pair of nodes.

In all the cases, we first need the following function (i.e., Algorithm 5) to do initialization. Then, at each time step, we update (i) the normalized adjacency matrices, $\mathbf{Mc}^{(t)}$ and $\mathbf{Mr}^{(t)}$, as well as the core matrix, $\mathbf{C}^{(t)}$; and we perform (ii) one or two sparse matrix–vector multiplications to get the proper answers. Compared to the update time (part (i)), the running time for part (ii) is always much less. So our algorithms can quickly give the proper answers at each time step. On the other hand,

Algorithm 5 Initialization

Input: The adjacency matrix at time step 1 $\mathbf{M}^{(1)}$, and the parameter c.
Output: The fixed degree matrix $\tilde{\mathbf{D}}$, the normalized matrices at time step 1 $\mathbf{Mr}^{(1)}$ and $\mathbf{Mc}^{(1)}$, and the initial core matrix $\mathbf{C}^{(\mathbf{1})}$.
1: get the fixed degree matrix $\tilde{\mathbf{D}}$ as equation (8.4)
2: normalize for type 1 objects: $\mathbf{Mr}^{(1)} = \mathbf{D}_1^{-1} \cdot \mathbf{M}^{(1)}$
3: normalize for type 2 objects: $\mathbf{Mc}^{(1)} = \mathbf{D}_2^{-1} \cdot \mathbf{M}'^{(1)}$
4: get the core matrix: $\mathbf{C}^{(\mathbf{1})} = (\mathbf{I} - c^2 \mathbf{Mc}^{(1)} \cdot \mathbf{Mr}^{(1)})^{-1}$
5: store the matrices: $\mathbf{Mr}^{(1)}$, $\mathbf{Mc}^{(1)}$, and $\mathbf{C}^{(1)}$.

we can easily verify that our algorithms give the exact answers, without any quality loss or approximation.

8.6.1 Track-Centrality

Here, we want to track the top-k most important type 1 (and/or type 2) nodes over time. For example, on an author–conference bipartite graph, we want to track the top-10 most influential authors (and/or conferences) over time. For a given query node, we also want to track its centrality (or the rank of centrality) over time. For example, on an author–conference bipartite graph, we can track the relative importance of an author in the entire community.

Based on the definition of centrality (8.3) and the fast update algorithms we developed in Section 8.4, we can get the following algorithm (Algorithm 6) to track the top-k queries over time. The algorithm for tracking centrality for a single query node is quite similar to Algorithm 6. We omit the details for space.

Algorithm 6 *Track-Centrality* (Top-k Queries)

Input: The time-evolving bipartite graphs $\{\mathbf{M}^{(1)}, \Delta\mathbf{M}^{(t)}(t \geq 2)\}$, the parameters c and k
Output: The top-k most central type 1 (and type 2) objects at each time step t.
1: **Initialization**
2: **for** each time step $t\,(t \geq 1)$ **do**
3: $x = \mathbf{1}_{1\times n} \cdot \mathbf{Mr}^{(t)} \cdot \mathbf{C}^{(\mathbf{t})}$; and $y = \mathbf{1}_{1\times l} \cdot \mathbf{C}^{(\mathbf{t})}$
4: $\mathbf{r_2}' = c \cdot x + y$
5: $\mathbf{r_1}' = c \cdot \mathbf{r_2'} \cdot \mathbf{Mc}^{(t)}$
6: output the top k type 1 objects according to $\mathbf{r_1}'$ (larger value means more central)
7: output the top k type 2 objects according to $\mathbf{r_2}'$ (larger value means more central)
8: Update $\mathbf{Mr}^{(t)}$, $\mathbf{Mc}^{(t)}$, and $\mathbf{C}^{(\mathbf{t})}$ for $t \geq 2$.
9: **end for**

In step 8 of Algorithm 6, we can either use *Fast-Single-Update* or *Fast-Batch-Update* to update the normalized matrices $\mathbf{Mr}^{(t)}$ and $\mathbf{Mc}^{(t)}$ and the core matrix $\mathbf{C}^{(\mathbf{t})}$. The running time for steps 3–8 is much less than the update time (step 8). Thus, *Track-Centrality* can give the ranking results quickly at each time step. On the other hand, using elementary linear algebra, we can easily prove the correctness of *Track-Centrality*:

Lemma 4 Correctness of Track-Centrality *The vectors* $\mathbf{r_1}'$ *and* $\mathbf{r_2}'$ *in Algorithm 6 provide a correct ranking of type 1 and type 2 objects at each time step t. That is, the ranking is exactly according to the centrality defined in* (8.3)*.*

Proof Based on Delta Matrix Inversion Theorems (theorem 4.2), we have that step 8 of *Track-Proximity* maintains the correct core matrix at each time step.

Apply the Sherman–Morrison lemma [25] to (8.2), we have

$$\mathbf{Q}^{(t)} \propto \begin{pmatrix} \mathbf{I} + c^2\mathbf{Mr}^{(t)}\mathbf{C^{(t)}}\mathbf{Mc}^{(t)} & c\mathbf{Mr}^{(t)}\mathbf{C^{(t)}} \\ c\mathbf{C^{(t)}}\mathbf{Mc}^{(t)} & \mathbf{C^{(t)}} \end{pmatrix}. \tag{8.9}$$

By (8.3), we have

$$\text{centrality}(j) \propto \sum_{i=1}^{n+l} r_{i,j}^{(t)} = \sum_{i=1}^{n+l} \mathbf{Q}^{(t)}(i, j).$$

Let $\mathbf{r} = [\text{centrality}(j)]_{j=1,\ldots,(n+l)}$, we have

$$\begin{aligned} \mathbf{r}' &\propto [\mathbf{1}_{1\times n},\ \mathbf{1}_{1\times l}] \cdot \mathbf{Q}^{(t)} \\ &\propto \begin{pmatrix} c^2\mathbf{1}_{1\times n}\mathbf{Mr}^{(t)}\mathbf{C^{(t)}}\mathbf{Mc}^{(t)} + c\mathbf{1}_{1\times l}\mathbf{C^{(t)}}\mathbf{Mc}^{(t)} \\ c\mathbf{1}_{1\times n}\mathbf{Mr}^{(t)}\mathbf{C^{(t)}} + \mathbf{1}_{1\times l}\mathbf{C^{(t)}} \end{pmatrix}', \\ &= \begin{pmatrix} c^2 x\mathbf{Mc}^{(t)} + cy\mathbf{Mc}^{(t)} \\ cx + y \end{pmatrix}' \\ &= [c\mathbf{r_2}'\mathbf{Mc}^{(t)},\ \mathbf{r_2}'] \\ &= [\mathbf{r_1}',\ \mathbf{r_2}'] \end{aligned}$$

where x and y are two vectors as defined in step 3 of *Track-Centrality* and $\mathbf{r_1}$, and $\mathbf{r_2}$ are two column vectors as defined in steps 4 and 5 of *Track-Centrality*. This completes the proof. □

8.6.2 Track-Proximity

Here, we want to track the top-k most related/relevant type 1 (and/or type 2) objects for object A at each time step. For example, on an author–conference bipartite graph evolving over time, we want track "*Which are the major conferences for John Smith in the past 5 year?*" or "*Who are most the related authors for John Smith so far?*" For a given pair of nodes, we also want to track their pairwise relationship over time. For example, in an author–conference bipartite graph evolving over time, we can track "*How much credit (a.k.a proximity) John Smith has accumulated in KDD?*"

Algorithm 7 *Track-Proximity* (Top-k Queries)

Input: The time-evolving bipartite graphs $\{\mathbf{M}^{(1)}, \Delta\mathbf{M}^{(t)}(t \geq 2)\}$, the parameters c and k, and the source node s.
Output: The top-k most related type 1 (and type 2) objects for s at each time step t.

Initialization
for each time step $t\,(t \geq 1)$ **do**
  for $i = 1 : n$ **do**
    $r_{s,i} = \mathbf{GetQij}(\mathbf{C}^{(t)}, \mathbf{Mr}^{(t)}, \mathbf{Mc}^{(t)}, s, i, c))$
  end for
  let $\mathbf{r_1} = [r_{s,i}](i = 1, ...n)$
  for $j = 1 : l$ **do**
    $r_{s,j} = \mathbf{GetQij}(\mathbf{C}^{(t)}, \mathbf{Mr}^{(t)}, \mathbf{Mc}^{(t)}, s, j + n, c))$
  end for
  let $\mathbf{r_2} = [r_{s,j}](j = 1, ...l)$
  output the top k type 1 objects according to $\mathbf{r_1}'$ (larger value means more relevant)
  output the top k type 2 objects according to $\mathbf{r_2}'$ (larger value means more relevant)
  update $\mathbf{Mr}^{(t)}$, $\mathbf{Mc}^{(t)}$, and $\mathbf{C^{(t)}}$ for $t \geq 2$.
end for

The algorithm for top-k queries is summarized in Algorithm 7. The algorithm for tracking the proximity for a given pair of nodes is quite similar to Algorithm 7. We omit its details for space.

In Algorithm 7, again, at each time step, the update time will dominate the total computational time. Thus by using either *Fast-Single-Update* or *Fast-Batch-Update*, we can quickly give the ranking results at each time step. Similar to *Track-Proximity*, we have the following lemma for the correctness of *Track-Proximity*:

Lemma 5 Correctness of Track-Proximity *The vectors* $\mathbf{r_1}'$ *and* $\mathbf{r_2}'$ *in Algorithm 7 provide a correct ranking of type 1 and type 2 objects at each time step* t. *That is, the ranking is exactly according to the proximity defined in* (8.4).

Proof Based on Delta Matrix Inversion Theorems (Theorem 4.2), we have that step 13 of *Track-Proximity* maintains the correct core matrix at each time step. Therefore, Algorithm 2 in step 8 always gives the correct proximity score, which completes the proof. □

8.7 Experimental Results

In this section we present experimental results, after we introduce the data sets in Section 8.6.1. All the experiments are designed to answer the following questions:

- *Effectiveness*: What is the quality of the applications (*Track-Centrality* and *Track-Proximity*) we proposed in this chapter?
- *Efficiency*: How fast are the proposed algorithms (*Fast-Single-Update* and *Fast-Batch-Update* for the update time, *Track-Centrality* and *Track-Proximity* for the overall running time)?

8.7.1 Data Sets

We use five different data sets in our experiments, summarized in Table 8.2. We verify the effectiveness of our proposed dynamic centrality measures on **NIPS**, **DM**, and **AC**, and measure the efficiency of our algorithms using the larger **ACPost** and **NetFlix** data sets.

Table 8.2 Data sets used in evaluations

Name	$n \times l$	Ave.$\hat{m}$	Time steps
NIPS	2,037×1,740	308	13
DM	5,095× 3,548	765	13
AC	418,236×3,571	26,508	49
ACPost	418,236×3,571	1,007	1258
NetFlix	2,649,429×17,770	100,480,507	NA

The first data set (**NIPS**) is from the NIPS proceedings.[2] The timestamps are publication years, so each graph slice **M** corresponds to 1 year, from 1987 to 1999. For each year, we have an author–paper bipartite graph. Rows represent authors and columns represent papers. Unweighted edges between authors and papers represent authorship. There are 2,037 authors, 1,740 papers, and 13 time steps (years) in total with an average of 308 new edges per year.

The **DM**, **AC**, and **ACPost** data sets are from DBLP.[3] For the first two, we use paper publication years as timestamps, similar to **NIPS**. Thus each graph slice **S** corresponds to 1 year.

DM uses author–paper information for each year between 1995 and 2007, from a restricted set of conferences, namely the five major data mining conferences ('KDD', 'ICDM', 'SDM', 'PKDD', and 'PAKDD'). Similar to **NIPS**, rows represent authors, columns represent papers, and unweighted edges between them represent authorship. There are 5,095 authors, 3,548 papers, and 13 time steps (years) in total, with an average of 765 new edges per time step.

AC uses author–conference information from the entire DBLP collection, between years 1959 and 2007. In contrast to **DM**, columns represent conferences and edges connect authors to conferences they have published in. Each edge in **S** is weighted by the number of papers published by the author in the corresponding conference for that year. There are 418,236 authors, 3,571 conferences, and 49 time steps (years) with an average of 26,508 new edges at each time step.

ACPost is primarily used to evaluate the scalability of our algorithms. In order to obtain a larger number of timestamps at a finer granularity, we use posting date on DBLP (the 'mdate' field in the XML archive of DBLP, which represents when the paper was entered into the database), rather than publication year. Thus, each graph slice **S** corresponds to 1 day, between January 3, 2002, and August 24, 2007. **ACPost** is otherwise similar to **AC**, with number of papers as edge weights. There

[2] http://www.cs.toronto.edu/∼roweis/data.html
[3] http://www.informatik.uni-trier.de/∼ley/db/

are 418,236 authors, 3,571 conferences, and 1,258 time steps (days with at least one addition into DBLP), with an average of 1,007 new edges per day.

The final data set, **NetFlix**, is from the Netflix prize.[4] Rows represent users and columns represent movies. If a user has rated a particular movie, we connect them with an unweighted edge. This data set consists of one slice and we use it in Section 8.6.2 to evaluate the efficiency *Fast-Single-Update*. In total, we have 2,649,429 users, 17,770 movies, and 100,480,507 edges.

8.7.2 Effectiveness: Case Studies

Here, we show the experimental results for the three applications on real data sets, all of which are consistent with our intuition.

8.7.2.1 Results on *Track-Centrality*

We apply Algorithm 6 to the **NIPS** data set. We use "Global Aggregation" to update the adjacency matrix $\mathbf{M}^{(t)}$. We track the top-k ($k = 10$) most central (i.e.influential) authors in the whole community. Table 8.3 lists the results for every 2 years. The results make sense: famous authors in the NIPS community show up in the top-10 list and their relative rankings change over time, reflecting their activity/influence in the whole NIPS community up to that year. For example, Prof. Terrence J. Sejnowski ('Sejnowski_T') shows in the top-10 list from 1989 on and his ranking keeps going up in the following years (1991, 1993). He remains number 1 from 1993 on. Sejnowski is one of the founders of NIPS, an IEEE Fellow, and the head of the Computational Neurobiology Lab at the Salk institute. The rest of the top-placed researchers include Prof. Michael I. Jordan ('Jordan_M') from UC Berkeley and Prof. Geoffrey E. Hinton ('Hinton_G') from University of Toronto, well known for their work in graphical models and neural networks, respectively. We can also track the centrality values as well as their rank for an individual author over the years. Figure 8.1 plots the centrality ranking for some authors over the years. Again, the

Table 8.3 Top-10 most influential (central) authors up to each year

1987	1989	1991	1993	1995	1997	1999
'Abbott_L'	'Bower_J'	'Hinton_G'	'Sejnowski_T'	'Sejnowski_T'	'Sejnowski_T'	'Sejnowski_T'
'Burr_D'	'Hinton_G'	'Koch_C'	'Koch_C'	'Jordan_M'	'Jordan_M'	'Koch_C'
'Denker_J'	'Tesauro_G'	'Bower_J'	'Hinton_G'	'Hinton_G'	'Koch_C'	'Jordan_M'
'Scofield_C	'Denker_J'	'Sejnowski_T'	'Mozer_M'	'Koch_C'	'Hinton_G'	'Hinton_G'
'Bower_J'	'Mead_C'	'LeCun_Y'	'LeCun_Y'	'Mozer_M'	'Mozer_M'	'Mozer_M'
'Brown_N'	'Tenorio_M'	'Mozer_M'	'Denker_J'	'Bengio_Y'	'Dayan_P'	'Dayan_P'
'Carley_L'	'Sejnowski_T'	'Denker_J'	'Bower_J'	'Lippmann_R'	'Bengio_Y'	'Singh_S'
'Chou_P'	'Lippmann_R'	'Waibel_A'	'Kawato_M'	'LeCun_Y'	'Barto_A'	'Bengio_Y'
'Chover_J'	'Touretzky_D'	'Moody_J'	'Waibel_A'	'Waibel_A'	'Tresp_V'	'Tresp_V'
'Eeckman_F'	'Koch_C'	'Lippmann_R'	'Simard_P'	'Simard_P'	'Moody_J'	'Moody_J'

[4] http://www.netflixprize.com/

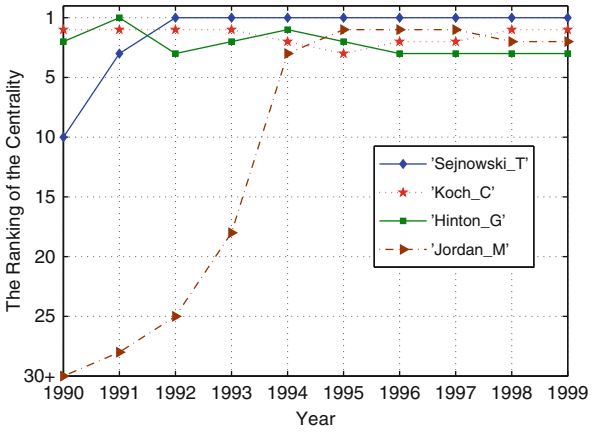


Fig. 8.1 The rank of centrality for some given authors in the whole **NIPS** data set up to each year

results are consistent with intuition. For example, Michael I. Jordon starts to have significant influence (within top-30) in the NIPS community from 1991 on; his influence rapidly increases in the following up years (1992–1995); and stays within the top-3 from 1996 on. Prof. Christof Koch ('Koch_C') from Caltech remains one of the most influential (within top-3) authors in the whole NIPS community over the years (1990–1999).

8.7.2.2 Results on *Track-Proximity*

We first report the results on the **DM** data set. We use "Global Aggregation" to update the adjacency matrix at each time step. In this setting, we can track the top-k most related papers/authors in the data mining community for a given query author up to each year. Table 8.4 lists the top-5 most related authors for 'Jian Pei' over the years (2001–2007). The results make perfect sense: (1) the top co-author (Prof. 'Jiawei Han') is Prof. Jian Pei's advisor; (2) the other top collaborators are either from SUNY-Buffalo (Prof. Aidong Zhang) or from IBM-Watson (Drs. Philip S. Yu, Haixun Wang, Wei Wang), which is also reasonable, since Prof. Pei held a faculty position at SUNY-Buffalo; (3) the IBM-Watson collaboration ('Philip S. Yu' and 'Haixun Wang') got stronger over time.

Table 8.4 Top-5 most related authors for 'Jian Pei' up to each year

2001	2003	2005	2007
'Jiawei_Han'	'Jiawei_Han'	'Jiawei_Han'	'Jiawei_Han'
'Behzad_Mortazavi-Asl'	'Behzad_Mortazavi-Asl'	'Haixun_Wang'	'Haixun_Wang'
'Hongjun_Lu'	'Aidong_Zhang'	'Aidong_Zhang'	'Philip_S._Yu'
'Meichun_Hsu'	'Philip_S._Yu'	'Philip_S._Yu'	'Wei_Wang'
'Shiwei_Tang'	'Hongjun_Lu'	'Wei_Wang'	'Aidong_Zhang'

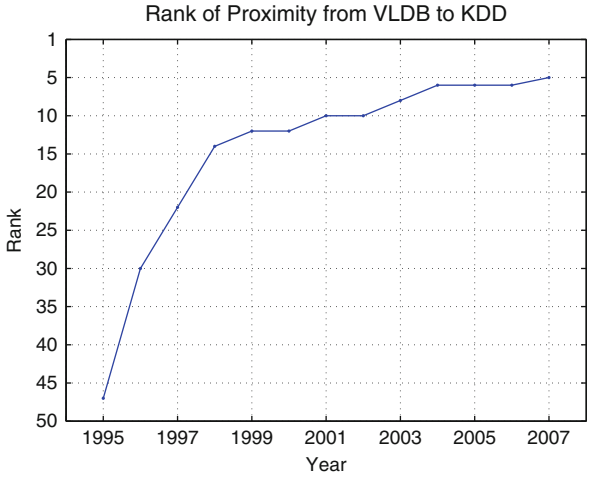


Fig. 8.2 The rank of the proximity from 'VLDB' to 'KDD' up to each year

Then, we apply *Track-Proximity* on the data set **AC**. Here, we want to track the proximity ranking for a given pair of nodes over time. Figure 8.2 plots the rank of proximity from the "VLDB" conference to the "KDD" conference. We use "Global Aggregation" to update the adjacency matrix. In this way, proximity between the "VLDB" and "KDD" conferences measures the importance/relevance of "KDD" wrt "VLDB" up to each year. From the figure, we can see that the rank of "KDD" keeps going up, reaching the fifth position by 2007. The other top-4 conferences for "VLDB" by 2007 are "SIGMOD," "ICDE," "PODS," and "EDBT", in this order. The result makes sense: with more and more multi-disciplinary authors publishing in both communities (databases and data mining), "KDD" becomes more and more closely related to "VLDB."

We also test the top-k queries on **AC**. Here, we use "Sliding Window" (with window length len $= 5$) to update the adjacency matrix. In this setting, we want to track the top-k most related conferences/authors for a given query node in the past 5 years at each time step t. Figure 8.3 lists the top-5 conferences for Dr. "Philip S. Yu." The major research interest (top-5 conferences) for "Philip S. Yu" is changing over time. For example, in the years 1988–1992, his major interest is in databases ("ICDE" and "VLDB"), performance ("SIGMETRICS"), and distributed systems ("ICDCS" and "PDIS"). However, during 2003–2007, while databases ("ICDE" and

ICDE	CIKM	KDD	ICDM
ICDCS	ICDCS	SIGMOD	KDD
SIGMETRICS	ICDE	ICDM	ICDE
PDIS	SIGMETRICS	CIKM	SDM
VLDB	ICMCS	ICDCS	VLDB
1992	1997	2002	2007

Fig. 8.3 Philip S. Yu's top five conferences at four time steps, using a window of 5 years

"VLDB") are still one of his major research interests, data mining became a strong research focus ("KDD," "SDM," and "ICDM").

8.7.3 *Efficiency*

After initialization, at each time step, most time is spent on updating the core matrix $\mathbf{C}^{(t)}$, as well as the normalized adjacency matrices. In this section, we first report the running time for update and then give the total running time for each time step. We use the two largest data sets (**ACPost** and **NetFlix**) to measure performance.

8.7.3.1 Update Time

We first evaluate *Fast-Single-Update*. Both **ACPost** and **NetFlix** are used. For each data set, we randomly add one new edge into the graph and compute the update time. The experiments are run multiple times. We compare *Fast-Single-Update* with Straight-Update (which does $l \times l$ matrix inversion at each time step) and the result is summarized in Fig. 8.4—Note that the y-axis is in log-scale). On both data sets, *Fast-Single-Update* requires significantly less computation: on **ACPost**, it is 64x faster (0.5 s vs. 32 s), while on **NetFlix**, it is 176x faster (22.5 s vs. 4, 313 s).

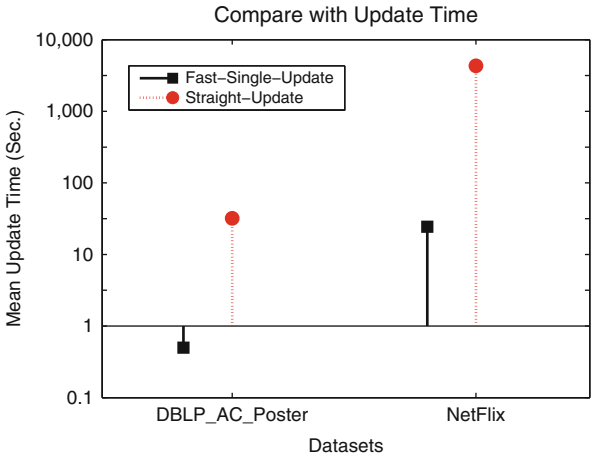
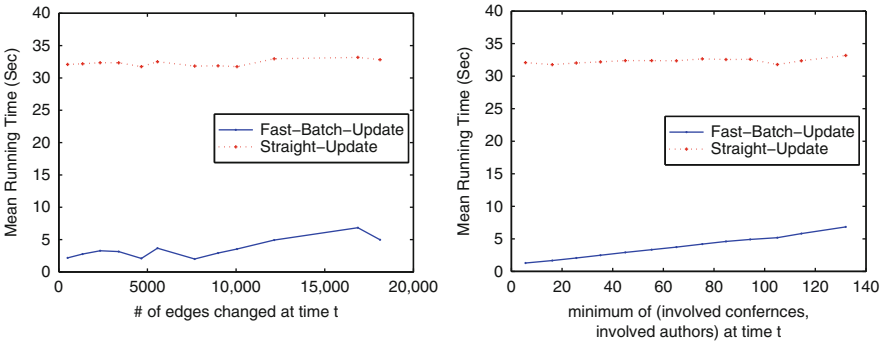


Fig. 8.4 Evaluation of *Fast-Single-Update*. For both data sets, one edge changes at each time step. The running time is averaged over multiple runs of experiments and shown in logarithmic scale

To evaluate *Fast-Batch-Update*, we use **ACPost**. From $t = 2$ and on, at each time step, we have between $\hat{m} = 1$ and $\hat{m} = 18{,}121$ edges updated. On average, there are 913 edges updated at each time step t ($t \geq 2$). Note that despite the large number of updated edges for some time steps, the rank of the difference matrix (i.e., $\min(\hat{n}, \hat{l})$) at each time step is relatively small, ranging from 1 to 132 with an



(a) Running Time vs. $\hat{m}$ (b) Running Time vs. $\min(\hat{n}, \hat{l})$

Fig. 8.5 Evaluation on *Fast-Batch-Update*

average of 33. The results are summarized in Fig 8.5. We plot the mean update time vs. the number ($\hat{m}$) of changed edges in Fig 8.5a and the mean update time vs. the rank ($\min(\hat{n}, \hat{l})$) of the update matrix in Fig 8.5b. Compared to the Straight-Update, *Fast-Batch-Update* is again much faster, achieving 5–32x speedup. On average, it is 15x faster than Straight-Update.

8.7.3.2 Total Running Time

Here, we study the total running time at each time step for *Track-Centrality*. The results for *Track-Proximity* are similar and omitted for space. For *Track-Centrality*, we let the algorithm return both the top-10 type 1 objects and the top-10 type 2 objects. We use the **NetFlix** data set with one edge changed at each time step and **ACPost** data set with multiple edges changed at each time step.

We compare our algorithms ("*Track-Centrality*") with (i) the one that uses Straight-Update in our algorithms (still referred as "Straight-Update") and (ii) that uses iterative procedure [27] to compute proximity and centrality at each time step (referred as "Ite-Alg"). The results are summarized in Fig. 8.6. We can see that in either case, our algorithm (*Track-Centrality*) is much faster. For example, it takes 27.8 s on average on the **NetFlix** data set, which is 155x faster over "Straight-Update" (4,315 s) and is 93x faster over "Ite-Alg" (2,582 s). In either case, the update time for *Track-Centrality* dominates the overall running time. For example, on the **ACPost** data set, update time accounts for more than 90% of the overall running time (2.4 s vs. 2.6 s). Thus, when we have to track queries for many nodes of interest, the advantage of *Track-Centrality* over "Ite-Alg" will be even more significant, since at each time step we only need to do update once for all queries, while the running time of "Ite-Alg" will increase linearly with respect to the number of queries.

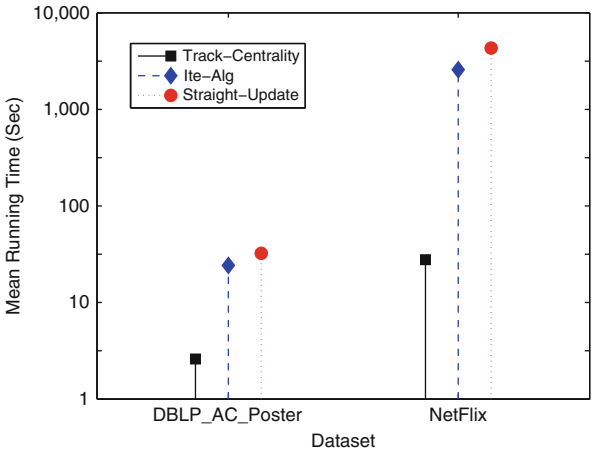


Fig. 8.6 Overall running time at each time step for *Track-Centrality*. For **ACPost**, there are multiple edges changed at each time step; and for **NetFlix**, there is only one edge changed at each time step. The running time is averaged in multiple runs of experiments and it is in the logarithm scale

8.8 Conclusion

In this chapter, we study how to incrementally track the node proximity as well as the centrality for time-evolving bipartite graphs. To the best of our knowledge, we are the first to study the node proximity and centrality in this setting. We first extend the proximity and centrality definitions to the setting of time-evolving graphs by degree-preserving operations. We then propose two fast update algorithms (*Fast-Single-Update* and *Fast-Batch-Update*) that do not resort to approximation and hence guarantee no quality loss (see Theorem 2), which are followed by two algorithms to incrementally track centrality (*Track-Centrality*) and proximity (*Track-Proximity*), in anytime fashion. We conduct extensive experimental case studies on several real data sets, showing how different queries can be answered, achieving up to **15∼176x** speedup. We can achieve such speedups while providing exact answers because we carefully leverage the fact that the rank of graph updates is small, compared to the size of the original matrix. Our experiments on real data show that this typically translates to at least an order of magnitude speedup.

Acknowledgments This material is based on the work supported by the National Science Foundation under Grants No. IIS-0705359 IIS0808661 and under the auspices of the US Department of Energy by University of California Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. This work is also partially supported by an IBM Faculty Award, a Yahoo Research Alliance Gift, a SPRINT gift, with additional funding from Intel, NTT, and Hewlett-Packard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

References

1. B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and S. S. Parag. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, Hong Kong, 2002.
2. R. Albert, H. Jeong, and A.-L. Barabasi. Diameter of the world wide web. *Nature*, 401: 130–131, 1999.
3. L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.
4. A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.
5. A. Broder, R. Kumar, F. Maghoul1, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: experiments and models. In *WWW Conf.*, 2000.
6. Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD*, pages 153–162, 2007.
7. S. Dorogovtsev and J. Mendes. Evolution of networks. *Advances in Physics*, 51:1079–1187, 2002.
8. C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.
9. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM*, pages 251–262, Aug-Sept. 1999.
10. G. Flake, S. Lawrence, C. L. Giles, and F. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3), Mar. 2002.
11. F. Geerts, H. Mannila, and E. Terzi. Relational link-based ranking. In *VLDB*, pages 552–563, 2004.
12. D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *9th ACM Conference on Hypertext and Hypermedia*, pages 225–234, New York, NY, 1998.
13. M. Girvan and M. E. J. Newman. Community structure is social and biological networks. In *PNAS*, pages 7821–7826, 2002.
14. M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In *Handbooks in Operations Research and Management Science 7: Network Models*. North Holland, 1993.
15. J. He, M. Li, H.-J. Zhang, H. Tong, and C. Zhang. Manifold-ranking based image retrieval. In *ACM Multimedia*, pages 9–16, 2004.
16. D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.
17. Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, 2006.
18. D. C. Kozen. *The Design and Analysis Algorithms*. Springer New York, NY 1992.
19. J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.
20. D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceeding of the 12th CIKM*, 2003.
21. E. Minkov and W. W. Cohen. An email and meeting assistant using graph walks. In *CEAS*, 2006.
22. M. E. J. Newman. The structure and function of complex networks. In *SIAM Review*, 45: 167–256, 2003.
23. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998. Paper SIDL-WP-1999-0120 (version of 11/11/1999).
24. J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.
25. W. Piegorsch and G. E. Casella. Inverting a sum of matrices. In *SIAM Review*, vol. 32, pages 470–470, 1990.

26. J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: Parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696, 2007.
27. J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.
28. J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: Dynamic tensor analysis. In *KDD*, pages 374–383, 2006.
29. H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD*, pages 404–413, 2006.
30. H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.
31. H. Tong, C. Faloutsos, and Y. Koren. Fast direction-aware proximity for graph mining. In *KDD*, pages 747–756, 2007.
32. H. Tong, C. Faloutsos, and J.-Y. Pan. Random walk with restart: Fast solutions and applications. *Knowledge and Information Systems: An International Journal (KAIS)*, 2007.
33. D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *VLDB*, pages 709–720, 2005.

Chapter 9
Discriminative Frequent Pattern-Based Graph Classification

Hong Cheng, Xifeng Yan, and Jiawei Han

Abstract Frequent graph mining has been studied extensively with many scalable graph mining algorithms developed in the past. Graph patterns are essential not only for exploratory graph mining but also for advanced graph analysis tasks such as graph indexing, graph clustering, and graph classification. In this chapter, we examine the frequent pattern-based classification of graph data. We will introduce different types of patterns used in graph classification and their efficient mining approaches. These approaches could directly mine the most discriminative subgraphs without enumerating the complete set of frequent graph patterns. The application of graph classification into chemical compound analysis and software behavior prediction will be discussed to demonstrate the power of discriminative subgraphs.

9.1 Introduction

With enormous amounts of graph data accumulated in scientific and commercial domains, such as biological networks, social networks, and software traces, there is an imminent need to develop scalable methods for the analysis of the overwhelmingly large and complex data. As an effective method for data analysis, classification has been widely used to categorize unlabeled data. Classification on graph data has many real applications, e.g., predicting whether a chemical compound has a desired biological activity or whether a software execution is erroneous, etc. Intuitively, the activity of a compound largely depends on its chemical structure and the arrangement of different atoms in the 3D space. Similarly, the correctness of a software execution largely depends on the control flow of the execution, which can be modeled as a flow graph. As a result, an effective classifier for graphs should be able to take into account their structural properties.

X. Yan (✉)
University of California at Santa Barbara, Santa Barbara, CA, USA
e-mail: xyan@cs.ucsb.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_9,

A lot of classification algorithms have been developed in the past, such as Decision Tree, Bayesian Network, and Support Vector Machine, most of which assume feature vector data. Unfortunately, graph data do not have a natural feature vector representation. Examples include chemical compounds, molecules, social and biological networks, and program flows. A primary question is how to construct discriminative and compact feature sets, on which classic vector space classifiers could be built to achieve good performance. Recent studies showed that graph pattern mining could be a promising approach for fitting graphs to vector data. Given a set of graphs, the discovered frequent graph patterns could be taken as features to project graphs in a vector space. For example, assume there is a training set $D = \{(G_1, y_1), \ldots, (G_n, y_n)\}$, where G_i is a graph and y_i is the class label. Let $\mathcal{F}_s = \{g_1, \ldots, g_m\}$ be the set of frequent patterns mined from D. Then a training graph $G_i \in D$ can be represented as a feature vector $\mathbf{x}(G_i) = [f(g_1, G_i), \ldots, f(g_m, G_i)]$, where $f(g_j, G_i)$ is the frequency of feature g_j in graph G_i. By transforming graphs to feature vectors, classifiers such as SVM can be applied directly.

In this chapter, we introduce a discriminative frequent pattern-based classification framework on graph data. We will first review related work [5, 7, 18, 19, 28, 30, 33] on classifying transactional data sets with discriminative frequent itemsets or association rules. Then we will describe different types of subgraph patterns [8, 12, 20, 29] as well as their usage in graph classification. Finally we will introduce several efficient mining approaches [9, 16, 25, 31] which directly mine the most discriminative subgraphs for classification. The application of graph classification techniques into real problems including chemical compound analysis [8] and software behavior prediction [4, 20] will also be discussed.

9.2 Problem Formulation

In graph classification, a training or test instance is represented as a labeled graph. A labeled graph has labels associated with its vertices and edges. The vertex set of a graph G is denoted by $V(G)$ and the edge set by $E(G)$. A label function, l, maps a vertex or an edge to a label.

Definition 1 (Graph Classification) Given a training set of graphs $D = \{G_i, y_i\}_{i=1}^n$, where $G_i \in \mathcal{G}$ is a labeled graph and $y_i \in \{\pm 1\}$ is the class label, graph classification is to induce a mapping $h(G) : \mathcal{G} \to \{\pm 1\}$ from D.

Subgraph isomorphism and *frequent subgraph* are defined as follows.

Definition 2 (Subgraph Isomorphism) For two labeled graphs g and g', a *subgraph isomorphism* is an injective function $f : V(g) \to V(g')$, s.t., (1), $\forall v \in V(g), l(v) = l'(f(v))$; and (2), $\forall (u, v) \in E(g), (f(u), f(v)) \in E(g')$ and $l(u, v) = l'(f(u), f(v))$, where l and l' are the labeling functions of g and g', respectively. f is called an embedding of g in g'.

Definition 3 (Frequent Subgraph) Given a labeled graph data set $D = \{G_1, \ldots, G_n\}$ and a subgraph g, the supporting graph set of g is $D_g = \{G_i | g \subseteq G_i, G_i \in D\}$. The support of g is $support(g) = \frac{|D_g|}{|D|}$. A frequent graph is a graph whose support is no less than a minimum support threshold, min_sup.

A graph g is a subgraph of another graph g' if there exists a subgraph isomorphism from g to g', denoted by $g \subseteq g'$. g' is called a supergraph of g.

9.3 Related Work

Pattern-based classification of graph data is related to *associative classification* on discretized transactional data, where association rules or itemsets are generated and analyzed for use in classification [5, 7, 18, 19, 28, 30, 33]. The basic idea is that strong associations between frequent itemsets and class labels can be discovered for classification. Prediction is made based on the top-ranked single rule or multiple rules.

CBA (Classification based on Associations) was the first associative classification method proposed by Liu et al. [19]. CBA consists of two parts, a rule generator and a classifier builder. The rule generator applies the Apriori algorithm [1] to generate a set of class association rules according to a *min_sup* threshold and a *min_conf* threshold. The class association rules are essentially association rules with a set of items at the rule left-hand side and the class attribute at the rule right-hand side. To build the best classifier out of a set of rules, one needs to evaluate all the possible subsets of rules on the training data and select the subset which gives the least number of errors. However, given m rules, there are 2^m subsets, which are computationally infeasible. A heuristic classifier builder is proposed in CBA. A total order is defined on the generated rules based on rule confidence and support. The classification rules are ranked according to the total order. Prediction is based on the first rule whose left-hand side condition satisfies the test case.

CMAR (Classification based on Multiple Association Rules) was proposed by Li et al. [18], which aims at improving the rule mining efficiency as well as avoiding bias or overfitting caused by the single rule-based classification. To improve mining efficiency, CMAR extends FP-growth and constructs a class distribution-associated FP-tree. Each node of the FP-tree registers not only the item attribute but also the class label distribution, for classification rule mining. To avoid classification bias and overfitting in CMAR, classification is performed based on a weighted χ^2 analysis on multiple high confidence, highly related rules.

CPAR (Classification based on Predictive Association Rules) was proposed by Yin and Han [33] which combine the advantages of associative classification and traditional rule-based classification. CPAR inherits the basic idea of FOIL [23] in rule generation. When selecting a literal to construct a rule, Foil Gain is used to measure the information gained from adding this literal to the current rule.

Instead of generating a large number of candidate rules, CPAR adopts a greedy algorithm to generate rules directly from training data with higher quality and

lower redundancy in comparison with associative classification. To avoid generating redundant rules, CPAR generates each rule by considering the set of "already-generated" rules. Compared with traditional rule-based classifiers, CPAR generates and tests more rules by considering all the close-to-the-best literals, instead of selecting only the best literal, so that important rules will not be missed. To avoid overfitting, CPAR uses expected accuracy to evaluate each rule and uses the best k rules in prediction.

Cong et al. [7] proposed to discover top-k covering rule groups for each row of gene expression profiles for classification purpose. Since the gene expression data are very high dimensional, they proposed a row enumeration technique and several pruning strategies to make the rule mining process very efficient. Experiments on real bioinformatics data sets show that the top-k covering rule mining algorithm is orders of magnitude faster than traditional association rule mining algorithms. A classifier RCBT is constructed from the top-k covering rule groups.

HARMONY [30] is another rule-based classifier which directly mines classification rules. It uses an instance-centric rule-generation approach to assure for each training instance that the highest confidence rule covering the instance is included in the rule set. Several search space pruning methods based on confidence and search strategies have been proposed, which can be pushed deeply into the rule discovery process. HARMONY is shown to be more efficient and scalable than previous rule-based classifiers.

Veloso et al. [28] proposed a lazy associative classification method, which performs computation on a demand-driven basis. Starting from a test instance, the lazy classifier projects the training data only on those features in the test instance. From the projected data set, association rules are discovered and ranked, and the top-ranked rule is used for prediction. This lazy classification method effectively reduces the number of rules produced by focusing on the test instance only.

Cheng et al. [5] provided some theoretical analysis to support the principle of frequent pattern-based classification. Reference [5] builds a connection between pattern frequency and discriminative measures, such as information gain and Fisher score, and shows that discriminative frequent patterns are essential for classification, whereas inclusion of infrequent patterns may not improve the classification accuracy due to their limited predictive power. A strategy is also proposed to set minimum support in frequent pattern mining for generating useful patterns. A follow-up study by Cheng et al. [6] proposed an efficient mining algorithm to directly generate the most discriminative frequent itemsets by accurately estimating the information gain upper bound and then integrating branch-and-bound search with FP-growth based on the estimated bound.

9.4 Mining Subgraph Features for Classification

In recent years, a lot of studies have been carried out to solve the graph classification problem. There are two major approaches: graph kernel-based and graph pattern-based approaches.

Graph kernel is an approach for graph classification without explicitly generating the subgraph patterns. It provides a way to measure the similarity between two graphs for classification, thus bypassing graph pattern mining. Many graph kernels have been proposed including random walk graph kernel [11, 14], optimal assignment kernel [10], shortest-path graph kernel [2], and subtree pattern kernel [24].

On the other hand, the graph pattern-based approach builds graph classifiers based on different types of graph substructure features. The basic idea is to extract frequent substructures [8, 15, 20], local graph fragments [29], or cyclic patterns and trees [12] and use them as descriptors to represent the graph data.

According to the performance comparison between graph kernel-based and graph pattern-based classification methods by [8, 31], the accuracy is comparable, but the graph kernel-based approach is much slower, since it needs to compute similarity of $O(n^2)$ pairs of graphs. The focus of this chapter is on the graph pattern-based classification approach. In the following, we will describe several types of graph patterns used as classification features. We will also discuss the feature selection issue in graph pattern-based classification.

9.4.1 Frequent Subgraph Features

Frequent subgraphs have been used as features for graph classification in [8, 15, 20]. Reference [8] presents a subgraph-based classification framework that decouples the frequent subgraph mining process from the classification model construction process. In the first step, frequent subgraph mining is applied to find all substructures present in the data set. Reference [8] proposed to use two types of subgraphs: *frequent topological subgraphs* (as defined in Definition 3) or *frequent geometric subgraphs* as graph features. The former only considers the 2D topology of a graph, whereas the latter considers a 3D coordinate for each vertex. While frequent topological subgraphs can be discovered by existing subgraph mining algorithms, e.g., [17], frequent geometric subgraph mining needs some special handling to incorporate the 3D structural information. To facilitate the discovery of frequent geometric subgraphs, a metric of average inter-atomic distance defined as the average Euclidean distance between all pairs of atoms in a molecule is used as a geometric signature of a graph. Hence a geometric subgraph consists of two components, a topological subgraph and an interval of average inter-atomic distance associated with it. A geometric graph contains this subgraph if it contains the topological subgraph and the average inter-atomic distance of the embedding is within the interval. The task of discovering geometric subgraphs now reduces to identifying those geometric configurations that are frequent enough and the interval of average inter-atomic distance is bounded by a tolerance threshold.

As the second step, feature selection is applied on top of the frequent subgraphs based on the sequential coverage paradigm [21]. The sequential covering algorithm takes as input a set of examples and the features discovered in these examples and iteratively applies the feature selection step. In each step, the algorithm selects

the feature that has the highest estimated accuracy. After selecting this feature, all the examples containing this feature are eliminated and the feature is marked as selected. The algorithm continues until either all the features are selected or all the examples are eliminated. After a set of discriminative frequent subgraph features are selected, the original labeled graphs can be represented in a vector format. Assume the feature set $\mathcal{F}_s = \{g_1, g_2, \ldots, g_m\}$ where each subgraph g_i represents a feature. Given a graph G and a feature g_i, $\mathbf{x}$ is the vector representing G. Then,

$$x_i = \begin{cases} 1, & g_i \text{ is a subgraph of } G, \\ 0, & \text{otherwise.} \end{cases} \tag{9.1}$$

Thus, $\mathbf{x}$ is a binary representation of graph G. One can also use the frequency of g_i in G as the feature value. This is referred as frequency representation. Finally, a classifier such as SVM is constructed on the feature vector representation of the graph data.

It is worth noting that frequent subgraph features are determined by the *min_sup* threshold. Therefore, the feature space can change if *min_sup* changes. An interesting problem is how to set *min_sup* to get the discriminative features.

A potential problem of using frequent substructures to represent graphs is the partial coverage problem, as pointed out in [29]. According to (9.1), if a graph G contains a subgraph feature g_i, its ith dimension has value 1 (or the number of occurrences of g_i in G), otherwise, 0. Given a set of frequent subgraphs $\mathcal{F}_s$ and a graph G, if none (or a small number) of the graph patterns in $\mathcal{F}_s$ is a subgraph of G, then the feature vector of G contains almost all 0s. That is, G is barely covered by the frequent subgraphs. This is referred as *partial feature coverage*. As a result, the topological property of such graphs is inadequately reflected by the frequent substructure features. It is usually difficult to distinguish such "uncovered" graphs from others.

9.4.1.1 Frequent Subgraph-Based Classification for Bug Localization

Liu et al. [20] modeled software executions as *software behavior graphs*, then built a frequent subgraph-based classification model to distinguish buggy executions from normal ones and used the classification result to further localize the bugs. A software behavior graph consists of a call graph and a transition graph. A call graph $G_c(\alpha)$ is a directed graph representing the function call relationship in a program execution α. The vertex set $V(G_c(\alpha))$ includes all the functions in α. An edge $(v_i, v_j) \in E(G_c(\alpha))$ if and only if function i calls function j in α. A transition graph $G_t(\alpha)$ is another directed graph displaying the function transition relationship in α. An edge $(v_i, v_j) \in E(G_t(\alpha))$ if and only if function j is called right after function i exits. The superposition of $G_c(\alpha)$ and $G_t(\alpha)$ forms the software behavior graph $G(\alpha)$ of α.

Similar with [8], Liu et al. [20] first applied closed subgraph mining [32] to generate subgraph features and then learned an SVM classifier using these features. Liu et al. used the graph classification technique to uncover "backtrace" for noncrashing

bugs based on the analysis of the classification accuracy boost. The basic idea is as follows. Generally, the classification accuracy should not decrease while more and more trace data become available; especially accuracy will improve once the execution data contain buggy behaviors. Suppose a program runs through components A, B, and C in sequence and a noncrashing bug resides in component B. Classifier f_A is trained at the end of execution of component A. As expected, its accuracy cannot be high since it knows few, if any, behaviors induced by the bug. In contrast, classifier f_B that is trained after component B is expected to have a much higher accuracy than f_A because it does have behavior graphs induced by the bug in incorrect runs. Therefore, as long as f_B has a classification accuracy boost in comparison with f_A, it is more likely that the bug is located in component B than component A.

Specifically, for each function, F_i, two checkpoints B^i_{in} and B^i_{out} are placed at the entrance and the exit of F_i, respectively. At each checkpoint, a classifier is trained using the traces running up to that point. A function with a very high accuracy boost (accuracy at B^i_{out} minus accuracy at B^i_{in}) is likely to be bug relevant.

9.4.2 Tree and Cyclic Pattern Features

Horváth et al. [12] proposed a method which represents a graph with a set of cyclic and tree patterns. The authors first define the set of cyclic patterns induced by the set of simple cycles of a graph. Let $G = (V, E, \Sigma, \lambda)$ be a graph where Σ is the set of labels and λ is the labeling function. Let $C = \{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_0\}$ be a sequence of edges that forms a simple cycle in G. The canonical representation of C is the lexicographically smallest string $\pi(C) \in \Sigma^*$ among the strings obtained by concatenating the labels along the vertices and edges of the cyclic permutations of C and its reverse. π is unique up to isomorphism, hence it indeed provides a canonical string representation of simple cycles. The set of cyclic patterns of a graph G, denoted by $\mathcal{C}(G)$ is defined by

$$\mathcal{C}(G) = \{\pi(C) : C \in \mathcal{S}(G)\},$$

where $\mathcal{S}(G)$ denotes the set of simple cycles of G.

Besides cyclic patterns, the method also considers the graph obtained by removing the edges of all simple cycles, or equivalently, by deleting every edge that belongs to some of G's biconnected components containing at least two edges. The resulting graph is a forest of trees consisting of the set of bridges of the graph. To assign a set of tree patterns to G, they use the forest formed by the set $\mathcal{B}(G)$ of bridges of G. Similar to simple cycles, they associate each tree T with a pattern $\pi(T) \in \Sigma^*$ that is unique to isomorphism and define the set of tree patterns $\mathcal{T}(G)$ assigned to G by

$$\mathcal{T}(G) = \{\pi(T) : \text{T is a connected component of } \mathcal{B}(G)\}.$$

The final feature space is the union of the cyclic patterns and tree patterns. The similarity of two graphs is measured by the cardinality of the intersection of their cyclic patterns plus the cardinality of the intersection of their tree patterns. A classifier is constructed on the training graphs based on the cyclic and tree pattern representation.

9.4.3 Graph Fragment Features

Wale and Karypis [29] proposed to use graph fragments to represent graphs. A graph fragment is a path, tree, or local graph structure enumerated from the graph data up to a finite length l in terms of the number of edges. Such enumerated graph fragments are ensured to have 100% coverage of the graph data and avoid the partial coverage problem by frequent subgraphs, since all graph fragments of length up to l are enumerated regardless of frequency. The graph fragments serve as classification features to represent a graph instance in a feature space. The recursive definition of graph fragments of length l is given by

$$F(G,l) = \begin{cases} \varnothing, \text{ if } G \text{ has fewer than } l \text{ edges or } l = 0, \\ eF(G\backslash e, l-1) \cup F(G\backslash e, l), \text{ otherwise.} \end{cases} \tag{9.2}$$

In addition, [29] provided a comparison between different types of graph structures in terms of four criteria: *topological complexity*, *generation process*, *precise representation*, and *complete coverage*. In terms of topological complexity, cyclic and tree (CT) patterns have medium complexity, while frequent subgraphs (FS) and graph fragments (GF) can have from low to high complexity, depending on the size of the patterns; for generation process, all three structures are dynamically generated from a given graph data set; all three structures have precise representation of a graph instance; in terms of coverage, CT and GF have complete coverage of a given graph data set while FS may have the partial coverage problem.

9.4.4 Feature Selection on Subgraph Patterns

When using frequent subgraphs as features, one usually faces a challenging problem that the number of frequent subgraphs may grow exponentially with the size of the training graphs, but only a few of them possess enough discriminative power to make them useful for graph classification. Therefore, an effective and efficient feature selection strategy is very critical in frequent subgraph-based classification. Previous frequent pattern-based classification studies use different heuristic strategies for feature selection. For example, [8] used sequential coverage paradigm [21], and [5] proposed MMRFS based on Maximal Marginal Relevance (MMR) [3] heuristic in information retrieval. Although these feature selection methods have been demonstrated to be effective in previous studies, they do not provide optimality guarantees.

A recent study by Thoma et al. [26] defined a principled, near-optimal approach to feature selection on frequent subgraphs. The method, called CORK, selects frequent subgraphs that greedily maximize a submodular quality criterion, thereby guaranteeing that the greedy solution to the feature selection problem is close to the globally optimal solution. Furthermore, the feature selection strategy can be integrated into gSpan, to help prune the search space for discriminative frequent subgraph mining.

The feature selection problem among frequent subgraphs can be cast as a combinatorial optimization problem. Let $\mathcal{F}$ denote the full set of subgraph features and $\mathcal{T}$ denote a subset of $\mathcal{F}$. Let $q(\mathcal{T})$ denote the relevance of a set of frequent subgraphs $\mathcal{T}$ for class membership, where q is a criterion measuring the discriminative power of $\mathcal{T}$. Feature selection can then be formulated as

$$\mathcal{F}_s = \arg\max_{\mathcal{T}\subseteq\mathcal{F}} q(\mathcal{T}) \quad s.t. \quad |\mathcal{T}| \leq s. \tag{9.3}$$

where $\mathcal{F}_s$ is the set of selected features, $|\cdot|$ computes the cardinality of a set and s is the maximally allowed number of selected features.

9.4.4.1 Submodularity

Assume that we measure the discriminative power $q(\mathcal{T})$ of a set of frequent subgraphs $\mathcal{T}$ in terms of a quality function q. A near-optimality solution is reached for a *submodular* quality function q when used in combination with greedy feature selection. Greedy forward feature selection iteratively picks a feature that, in union with the features selected so far, maximizes the quality function q over the perspective feature set. In general, this strategy will not yield an optimal solution, but it can be shown to yield a near-optimal solution if q is submodular.

Definition 4 (Submodular Set Function) A quality function q is said to be submodular on a set $\mathcal{F}$ if for $\mathcal{T}' \subset \mathcal{T} \subseteq \mathcal{F}$ and $X \in \mathcal{F}$,

$$q(\mathcal{T}' \cup \{X\}) - q(\mathcal{T}') \geq q(\mathcal{T} \cup \{X\}) - q(\mathcal{T}). \tag{9.4}$$

If q is submodular and we employ greedy forward feature selection, then there is a theorem from [22] that holds.

Theorem 1 *If q is a submodular, nondecreasing set function on a set $\mathcal{F}$ and $q(\phi) = 0$, then greedy forward feature selection is guaranteed to find a set of feature $\mathcal{T} \subseteq \mathcal{F}$ such that*

$$q(\mathcal{T}) \geq \left(1 - \frac{1}{e}\right) \max_{\mathcal{U}\subseteq\mathcal{F}:|\mathcal{U}|=s} q(\mathcal{U}), \tag{9.5}$$

where s is the number of features to be selected.

As a consequence, the result from greedy feature selection achieves at least $(1 - \frac{1}{e}) \approx 63\%$ of the score of the optimal solution to the feature selection problem. This is referred to as being *near-optimal* in the literature.

9.4.4.2 The Proposed Quality Criterion CORK

Definition 5 (Correspondence) A pair of data objects $(v^{(i)}, v^{(j)})$ is called a correspondence in a set of features indicated by indices $\mathcal{U} \subseteq \{1, \ldots, |\mathcal{F}|\}$ iff

$$(v^{(i)} \in D_+) \wedge (v^{(j)} \in D_-) \wedge \forall d \in \mathcal{U} : (v_d^{(i)} = v_d^{(j)}), \tag{9.6}$$

where D_+ is the set of positive training graphs and D_- is the set of negative training graphs, and $\mathcal{F}$ is the set of subgraph features.

Definition 6 (CORK) The proposed quality criterion q is called CORK (Correspondence-based Quality Criterion) for a subset of features $\mathcal{U}$ as

$$q(\mathcal{U}) = (-1) \times \text{number of correspondences in } \mathcal{U}. \tag{9.7}$$

Reference [26] provided a theorem to show that CORK is submodular. Then CORK can be used for greedy forward feature selection on a pre-mined set $\mathcal{F}$ of frequent subgraphs from G and get a result set $\mathcal{F}_s \subseteq \mathcal{F}$ with a guaranteed quality bound.

9.4.4.3 Integrating CORK with gSpan to Prune Search Space

Furthermore, [26] showed that CORK can be integrated into gSpan to directly mine a set of discriminative subgraph features. The basic idea is, from the CORK-value of a subgraph g, an upper bound for the CORK-values of all its supergraphs can be derived, thus allowing to prune the search space. Theorem 2 shows the upper bound of the CORK-values for supergraphs of a given subgraph g, which is useful in a branch-and-bound search for pruning search space in gSpan.

Theorem 2 *Let g be a subgraph and g' is a supergraph of g. Let $D_+(g^{(1)})$ denote the set of graphs in the positive class that contains g, $D_+(g^{(0)})$ denote the set of graphs in the positive class that does not contain g. Define $D_-(g^{(1)})$ and $D_-(g^{(0)})$ analogously. Then*

$$q(\{g\}) = -(|D_+(g^{(0)})| * |D_-(g^{(0)})| + |D_+(g^{(1)})| * |D_-(g^{(1)})|) \tag{9.8}$$

and

$$q(\{g'\}) \leq q(\{g\}) + \max \left\{ \begin{array}{l} |D_+(g^{(1)})| \cdot (|D_-(g^{(1)})| - |D_-(g^{(0)})|) \\ (|D_+(g^{(1)})| - |D_+(g^{(0)})|) \cdot |D_-(g^{(1)})| \\ 0 \end{array} \right\}. \tag{9.9}$$

The bound in (9.9) can be extended to an upper bound over a set of selected features to estimate the overall quality of the feature set.

9.5 Direct Mining Strategies

Many of the above introduced graph classification methods [8, 12, 20, 29] first mine the complete set of graph substructures $\mathcal{F}$ wrt. a minimum support threshold or other parameters such as subgraph size limit, then use a feature selection algorithm to select a small set of discriminative subgraphs as features for classification. The mining step is "uninformed" in terms of the discriminative score of generated subgraphs. Therefore, it usually generates a much larger set of subgraphs than what is actually used for classification, making this method very computationally expensive. If the input graph data set is very large or the minimum support threshold is low, the mining step may not even be completed due to the combinatorial explosion. As opposed to this approach, recent studies have focused on an efficient mining approach which directly generates a compact set of discriminative subgraphs, while avoiding generating low discriminative subgraphs. These studies usually use an objective function to evaluate the quality of a subgraph, then search for the best subgraph in the subgraph pattern space with different pruning strategies or models. The CORK-based method, described in Section 9.4.4.3, is such an example. In this section, we will introduce the mining techniques of four recent studies: gboost [16], LEAP [31], GraphSig [25], and M^{b}T [9].

9.5.1 gboost: A Branch-and-Bound Approach

Kudo, Maeda and Matsumoto [16] presented an application of boosting for classifying labeled graphs, such as chemical compounds, natural language texts. A weak classifier called decision stump uses a subgraph as a classification feature. Then a boosting algorithm repeatedly constructs multiple weak classifiers on weighted training instances. A gain function is designed to evaluate the quality of a decision stump, i.e., how many weighted training instances can be correctly classified. Then the problem of finding the optimal decision stump in each iteration is formulated as mining an "optimal" subgraph pattern. gboost designs a branch-and-bound mining approach based on the gain function and integrates it into gSpan to search for the "optimal" subgraph pattern. Tsuda [27] proposed a similar pattern selection approach, but using regularization.

9.5.1.1 A Boosting Framework

gboost uses a simple classifier, *decision stump*, for prediction according to a single feature. The subgraph-based decision stump is defined as follows.

Definition 7 (Decision Stumps for Graphs) Let t and $\mathbf{x}$ be labeled graphs and $y \in \{\pm 1\}$ be a class label. A decision stump classifier for graphs is given by

$$h_{\langle t,y\rangle}(\mathbf{x}) = \begin{cases} y, & t \subseteq \mathbf{x}, \\ -y, & \text{otherwise.} \end{cases}$$

The decision stumps are trained to find a rule $\langle \hat{t}, \hat{y} \rangle$ that minimizes the error rate for the given training data $T = \{\langle \mathbf{x}_i, y_i \rangle\}_{i=1}^{L}$:

$$\begin{aligned} \langle \hat{t}, \hat{y} \rangle &= \arg \min_{t \in \mathcal{F}, y \in \{\pm 1\}} \frac{1}{L} \sum_{i=1}^{L} I(y_i \neq h_{\langle t,y\rangle}(\mathbf{x}_i)) \\ &= \arg \min_{t \in \mathcal{F}, y \in \{\pm 1\}} \frac{1}{2L} \sum_{i=1}^{L} (1 - y_i h_{\langle t,y\rangle}(\mathbf{x}_i)). \end{aligned} \tag{9.10}$$

where $\mathcal{F}$ is a set of graph features (i.e., $\mathcal{F} = \bigcup_{i=1}^{L} \{t | t \subseteq \mathbf{x}_i\}$) and $I(\cdot)$ is an indicator function. The gain function for a rule $\langle t, y \rangle$ is defined as

$$\text{gain}(\langle t, y \rangle) = \sum_{i=1}^{L} y_i h_{\langle t,y\rangle}(\mathbf{x}_i). \tag{9.11}$$

Using the gain, the search problem in (9.10) becomes equivalent to the problem: $\langle \hat{t}, \hat{y} \rangle = \arg\max_{t \in \mathcal{F}, y \in \{\pm 1\}} \text{gain}(\langle t, y \rangle)$. Then the gain function is used instead of error rate.

gboost then applies AdaBoost by repeatedly calling the decision stumps and finally produces a hypothesis f, which is a linear combination of K hypotheses produced by the decision stumps $f(\mathbf{x}) = \text{sgn}(\sum_{k=1}^{K} \alpha_k h_{\langle t_k, y_k \rangle}(\mathbf{x}))$. In the kth iteration, a decision stump is built with weights $\mathbf{d}^{(k)} = (d_1^{(k)}, \ldots, d_L^{(k)})$ on the training data, where $\sum_{i=1}^{L} d_i^{(k)} = 1$, $d_i^{(k)} \geq 0$. The weights are calculated to concentrate more on hard examples than easy ones. In the boosting framework, the gain function is redefined as follows:

$$\text{gain}(\langle t, y \rangle) = \sum_{i=1}^{L} y_i d_i h_{\langle t,y\rangle}(\mathbf{x}_i). \tag{9.12}$$

9.5.1.2 A Branch-and-Bound Search Approach

According to the gain function in (9.12), the problem of finding the optimal rule $\langle \hat{t}, \hat{y} \rangle$ from the training data set is defined as follows.

Problem 1 [Find Optimal Rule] Let $T = \{\langle \mathbf{x}_1, y_1, d_1 \rangle, \ldots, \langle \mathbf{x}_L, y_L, d_L \rangle\}$ be training data where $\mathbf{x}_i$ is a labeled graph, $y_i \in \{\pm 1\}$ is a class label associated with $\mathbf{x}_i$ and d_i ($\sum_{i=1}^{L} d_i = 1$, $d_i \geq 0$) is a normalized weight assigned to $\mathbf{x}_i$. Given T, find the optimal rule $\langle \hat{t}, \hat{y} \rangle$ that maximizes the gain, i.e., $\langle \hat{t}, \hat{y} \rangle = \arg\max_{t \in \mathcal{F}, y \in \{\pm 1\}} \sum_{i=1}^{L} y_i d_i h_{\langle t,y\rangle}$, where $\mathcal{F} = \bigcup_{i=1}^{L} \{t | t \subseteq \mathbf{x}_i\}$.

A naive method is to enumerate all subgraphs $\mathcal{F}$ and then calculate the gains for all subgraphs. However, this method is impractical since the number of subgraphs is exponential to its size. To avoid such exhaustive enumeration, a branch-and-bound algorithm is developed based on the following upper bound of the gain function.

Lemma 1 (Upper bound of the gain) *For any $t' \supseteq t$ and $y \in \{\pm 1\}$, the gain of $\langle t', y\rangle$ is bounded by $\mu(t)$ (i.e., $gain(\langle t', y\rangle) \leq \mu(t)$), where $\mu(t)$ is given by*

$$\mu(t) = \max\left(2 \sum_{\{i|y_i=+1, t\subseteq x_i\}} d_i - \sum_{i=1}^{L} y_i \cdot d_i, 2 \sum_{\{i|y_i=-1, t\subseteq x_i\}} d_i + \sum_{i=1}^{L} y_i \cdot d_i\right). \tag{9.13}$$

Figure 9.1 depicts a graph pattern search tree where each node represents a graph. A graph g' is a child of another graph g if g' is a supergraph of g with one more edge. g' is also written as $g' = g \diamond e$, where e is the extra edge. In order to find the optimal rule, the branch-and-bound search in the search tree estimates the upper bound of the gain function for all descendants below a node g. If it is smaller than the value of the best subgraph seen so far, it cuts the search branch of that node. Under the branch-and-bound search, a tighter upper bound is always preferred since it means faster pruning.

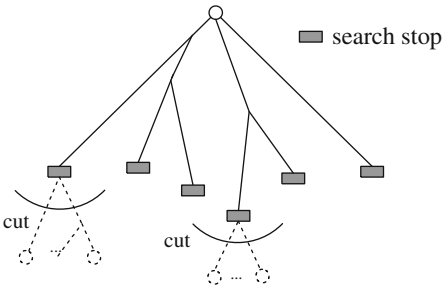


Fig. 9.1 Branch-and-bound search

Algorithm 1 outlines the framework of branch-and-bound for searching the optimal graph pattern. In the initialization, all the subgraphs with one edge are enumerated first and these seed graphs are then iteratively extended to large subgraphs. Since the same graph could be grown in different ways, line 5 checks whether it has been discovered before; if it has, then there is no need to grow it again. The optimal gain($\langle \hat{t}, \hat{y}\rangle$) among all the previously calculated gains is maintained. If $\mu(t) \leq$ gain($\langle \hat{t}, \hat{y}\rangle$), the gain of any supergraph $t' \supseteq t$ is no greater than gain($\langle \hat{t}, \hat{y}\rangle$), and therefore the branch can safely be pruned.

Algorithm 1 Branch-and-bound

Input: Graph dataset D
Output: Optimal rule $\langle \hat{t}, \hat{y} \rangle$

1: $S = \{1\text{-edge graph}\}$;
2: $\langle \hat{t}, \hat{y} \rangle = \varnothing$; $gain(\langle \hat{t}, \hat{y} \rangle) = -\infty$;
3: **while** $S \neq \varnothing$ **do**
4: choose t from S, $S = S \setminus \{t\}$;
5: **if** t was examined **then**
6: **continue**;
7: **if** $gain(\langle t, y \rangle) > gain(\langle \hat{t}, \hat{y} \rangle)$ **then**
8: $\langle \hat{t}, \hat{y} \rangle = \langle t, y \rangle$;
9: **if** $\mu(t) \leq gain(\langle \hat{t}, \hat{y} \rangle)$ **then**
10: **continue**;
11: $S = S \cup \{t' | t' = t \diamond e\}$;
12: **return** $\langle \hat{t}, \hat{y} \rangle$;

9.5.2 LEAP: A Structural Leap Search Approach

Yan et al. [31] proposed an efficient algorithm which mines the most significant subgraph pattern with respect to an objective function. A major contribution of this study is the proposal of a general approach for significant graph pattern mining with non-monotonic objective functions. The mining strategy, called LEAP (Descending Leap Mine), explored two new mining concepts: (1) *structural leap search* and (2) *frequency-descending mining*, both of which are related to specific properties in pattern search space. We will describe the first strategy as well as the application of this mining method to graph classification. Interested readers can refer to [31] for more details.

9.5.2.1 Structural Leap Search

LEAP assumes each input graph is assigned either a positive or a negative label (e.g., compounds active or inactive to a virus). One can divide the graph data set into two subsets: a positive set D_+ and a negative set D_-. Let $p(g)$ and $q(g)$ be the frequency of a graph pattern g in positive graphs and negative graphs. Many objective functions can be represented as a function of p and q for a subgraph pattern g, as $F(g) = f(p(g), q(g))$.

Figure 9.2 shows a search space of subgraph patterns. If we examine the search structure horizontally, we find that the subgraphs along the neighbor branches likely have similar compositions and frequencies, hence similar objective score. Take the branches A and B as an example. Suppose A and B split on a common subgraph pattern g. Branch A contains all the supergraphs of $g \diamond e$ and B contains all the supergraphs of g except those of $g \diamond e$. For a graph g' in branch B, let $g'' = g' \diamond e$ in branch A.

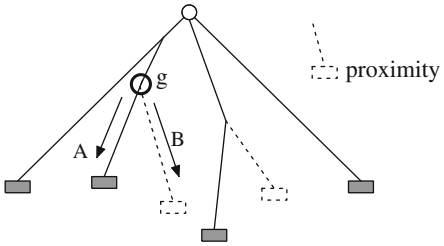


Fig. 9.2 Structural proximity

If in a graph data set, $g \diamond e$ and g often occur together, then g'' and g' might also often occur together. Hence, likely $p(g'') \sim p(g')$ and $q(g'') \sim q(g')$, which means similar objective scores. This is resulted by the structural similarity and embedding similarity between the starting structures $g \diamond e$ and g. We call it *structural proximity*: Neighbor branches in the pattern search tree exhibit strong similarity not only in pattern composition, but also in their embeddings in the graph data sets, thus having similar frequencies and objective scores. In summary, a conceptual claim can be drawn,

$$g' \sim g'' \Rightarrow F(g') \sim F(g''). \tag{9.14}$$

According to structural proximity, it seems reasonable to skip the whole search branch once its nearby branch is searched, since the best scores between neighbor branches are likely similar. Here, we would like to emphasize "likely" rather than "surely." Based on this intuition, if the branch A in Fig. 9.2 has been searched, B could be "leaped over" if A and B branches satisfy some similarity criterion. The length of leap can be controlled by the frequency difference between two graphs g and $g \diamond e$. The leap condition is defined as follows.

Let $I(G, g, g \diamond e)$ be an indicator function of a graph G: $I(G, g, g \diamond e) = 1$, for any supergraph g' of g, if $g' \subseteq G$, $\exists g'' = g' \diamond e$ such that $g'' \subseteq G$; otherwise 0. When $I(G, g, g \diamond e) = 1$, it means if a supergraph g' of g has an embedding in G, there must be an embedding of $g' \diamond e$ in G. For a positive data set D_+, let $D_+(g, g \diamond e) = \{G | I(G, g, g \diamond e) = 1, g \subseteq G, G \in D_+\}$. In $D_+(g, g \diamond e)$, $g' \supset g$ and $g'' = g' \diamond e$ have the same frequency. Define $\Delta_+(g, g \diamond e)$ as follows,

$$\Delta_+(g, g \diamond e) = p(g) - \frac{|D_+(g, g \diamond e)|}{|D_+|}.$$

$\Delta_+(g, g \diamond e)$ is actually the maximum frequency difference that g' and g'' could have in D_+. If the difference is smaller than a threshold σ, then leap,

$$\frac{2\Delta_+(g, g \diamond e)}{p(g \diamond e) + p(g)} \leq \sigma \text{ and } \frac{2\Delta_-(g, g \diamond e)}{q(g \diamond e) + q(g)} \leq \sigma. \tag{9.15}$$

σ controls the leap length. The larger σ is, the faster the search is. Structural leap search will generate an optimal pattern candidate and reduce the need for thoroughly searching similar branches in the pattern search tree. Its goal is to help a program search significantly distinct branches and limit the chance of missing the most significant pattern. Different from the widely used branch-and-bound search strategy which prunes the search space "vertically," structural leap search identifies similarly sibling branches and prunes the search space "horizontally." It can prune more search space and thus is more efficient. However, it may miss the globally optimal subgraph while branch-and-bound is guaranteed to find the best one.

Algorithm 2 outlines the pseudocode of structural leap search. The leap condition is tested on lines 7–8.

Algorithm 2 Structural leap search: sLeap(D, σ, $g^\star$)

Input: Graph dataset D, difference threshold σ
Output: Optimal graph pattern candidate $g^\star$

1: $S = \{1 - \text{edge graph}\}$;
2: $g^\star = \varnothing$; $F(g^\star) = -\infty$;
3: **while** $S \neq \varnothing$ **do**
4: $S = S \setminus \{g\}$;
5: **if** g was examined **then**
6: **continue**;
7: **if** $\exists g \diamond e, g \diamond e \prec g, \frac{2\Delta_+(g, g\diamond e)}{p(g\diamond e)+p(g)} \leq \sigma, \frac{2\Delta_-(g, g\diamond e)}{q(g\diamond e)+q(g)} \leq \sigma$
8: **continue**;
9: **if** $F(g) > F(g^\star)$ **then**
10: $g^\star = g$;
11: **if** $\widehat{F}(g) \leq F(g^\star)$ **then**
12: **continue**;
13: $S = S \cup \{g' | g' = g \diamond e\}$;
14: **return** $g^\star$;

9.5.2.2 Mining Discriminative Patterns for Graph Classification

To mine discriminative subgraphs for graph classification, information gain is used as the objective function in LEAP. As LEAP only generates the single best subgraph in a graph data set while multiple features are needed for classification, Yan et al. adopted the concept of sequential coverage paradigm [21] to generate multiple features to achieve a proper coverage on the training data set. It applies LEAP iteratively to generate multiple features to cover the training graphs: given a training graph data set D and an objective function of information gain, call LEAP to find the subgraph g_1 with the highest information gain. Then all the training graphs containing g_1, i.e., $TS(g_1) \in D$, are removed from further consideration. On the remaining training set, repeat this process until D becomes empty. Algorithm 3 outlines the graph feature mining process.

Algorithm 3 Mining discriminative graph features with LEAP

Input: Graph dataset D, objective function F
Output: A feature set $\mathcal{F}_s = \{g_1, \ldots, g_m\}$

1: $\mathcal{F}_s = \varnothing, i = 1$;
2: **while** D is not empty
3: $g_i = LEAP(D, F)$;
4: $\mathcal{F}_s = \mathcal{F}_s \cup \{g_i\}$;
5: $D = D - TS(g_i)$;
6: i++;
7: **end while**
8: **return** $\mathcal{F}_s$;

9.5.2.3 Mining Discriminative Subgraphs for Bug Localization

Cheng et al. [4] formulated the bug localization problem as a discriminative graph mining problem: given software behavior graphs (defined similarly as [20] that is introduced in Section 9.4.1.1) of correct executions and faulty executions, apply LEAP to extract the most discriminative subgraphs which contrast the program flow of correct and faulty executions. Intuitively, a bug will cause structural differences between the software behavior graphs of faulty and correct runs. Different from traditional bug localization methods pinpointing a single line of code which is likely to contain bugs, the discriminative subgraphs serve as signatures highlighting potential bug locations and provide informative contexts where bugs occur, which in turn help to guide programmers in finding the source of bugs and correcting them. Formally, the problem is defined in the LEAP framework as follows.

Definition 8 (Mining Optimal Bug Signature) Given a set of graphs with class labels, $D = \{G_i, y_i\}_{i=1}^n$, where $G_i \in \mathcal{G}$ is a software behavior graph representing an execution and $y_i \in \{\pm 1\}$ is the class label representing a correct or faulty status, an objective function F, find a subgraph g^* such that $g^* = \text{argmax}_g F(g)$.

Discriminative measures such as information gain, cross entropy, and Fisher score are popularly used to evaluate the capacity of a feature in distinguishing instances from different classes. In this work, Cheng et al. use information gain as the objective function. According to information gain, if the frequency difference of a subgraph in the faulty executions and the correct executions increases, the subgraph becomes more discriminative. A subgraph which occurs frequently in faulty executions but rarely in correct executions will have a very large information gain score and indicate that the corresponding partial software behavior graph is very discriminative to differentiate faulty executions from correct ones.

In the bug localization problem, it does not suffice to report the single best location only. Rather, it would be more informative to generate a ranked list of discriminative subgraphs with descending scores which are highly indicative of bugs and their contexts. To handle such requirements, [4] propose an extension from LEAP to mine top-k discriminative subgraphs. Formally, the problem is defined as follows:

Definition 9 (Mining Top-K Bug Signatures) Given a set of software behavior graphs with class labels, $D = \{G_i, y_i\}_{i=1}^{n}$, an objective function F, find k subgraphs $G_k = \{g_i^*\}_{i=1}^{k}$ from D which maximize $\sum_{i=1}^{k} F(g_i^*)$.

Accordingly, the LEAP algorithm will be modified to generate k discriminative subgraphs through k iterations. Initially LEAP finds the most discriminative subgraph g_1^* from D wrt. the objective function F and inserts it into G_k. It iteratively finds a subgraph g_i^* with the highest F function score which is also different from all existing subgraphs in G_k. Such a subgraph g_i^* is inserted into G_k and this process is iterated for k times. Finally, G_k is returned. If there exist less than k subgraphs from D, the search process will terminate early. The extended algorithm is called Top-K LEAP, as shown in Algorithm 4.

Algorithm 4 Mining top-K discriminative graph patterns top-K LEAP(D, F, k, G_k)

Input: Graph dataset D, objective function F, subgraph number k
Output: Top-k discriminative subgraphs G_k

```
1: G_k = ∅;
2: for i from 1 to k
3:     g_i^* = LEAP(F, G_k);
4:     if g_i^* = ∅
5:         break;
6:     G_k = G_k ∪ {g_i^*};
7: end for
8: return G_k;
```

Experimental studies on the Siemens benchmark data sets are performed to evaluate the proposed subgraph mining method. Experimental results demonstrate that it is effective in recovering bugs and their contexts. On average, it improves the precision and recall of a recent study RAPID [13] by up to 18.1 and 32.6%, respectively. In addition, the method is shown to be more efficient. It is able to complete subgraph mining on every Siemens data set within 258 s while RAPID is not able to complete mining some data sets even after running for hours.

9.5.3 GraphSig: A Feature Representation Approach

Ranu and Singh [25] proposed GraphSig, a scalable method to mine significant (measured by p-value) subgraphs based on a feature vector representation of graphs. The significant subgraphs can be used as classification features, since they are able to describe a property where the data set deviates from expected. The first step in the mining process is to convert each graph into a set of feature vectors where each vector represents a region within the graph. Prior probabilities of features are computed empirically to evaluate statistical significance of patterns in the feature space. Following the analysis in the feature space, only a small portion of the

exponential search space is accessed for further analysis. This enables the use of existing frequent subgraph mining techniques to mine significant patterns in a scalable manner even when they are infrequent. The major steps of GraphSig are described as follows.

9.5.3.1 Sliding Window Across Graphs

As the first step, random walk with restart (abbr. RWR) is performed on each node in a graph to simulate sliding a window across the graph. RWR simulates the trajectory of a random walker that starts from the target node and jumps from one node to a neighbor. Each neighbor has an equal probability of becoming the new station of the walker. At each jump, the feature traversed is updated which can either be an edge type or atom type. A restart probability α brings the walker back to the starting node within approximately $\frac{1}{\alpha}$ jumps. The random walk iterates till the feature distribution converges. As a result, RWR produces a continuous distribution of features for each node where a feature value lies in the range [0, 1], which is further discretized into 10 bins. RWR can therefore be visualized as placing a window at each node of a graph and capturing a feature vector representation of the subgraph within it. A graph of m nodes is represented by m feature vectors. RWR inherently takes proximity of features into account and preserves more structural information than simply counting the occurrence of features inside the window.

9.5.3.2 Calculating p-Value of A Feature Vector

To calculate p-value of a feature vector, we model the occurrence of a feature vector $\underline{x}$ in a random feature vector generated from a random graph. The frequency distribution of a vector is generated using the prior probabilities of features obtained empirically. Given a feature vector $\underline{x} = [x_1, \ldots, x_n]$, the probability of $\underline{x}$ occurring in a random feature vector $\underline{y} = [y_1, \ldots, y_n]$ can be expressed as a joint probability

$$P(\underline{x}) = P(y_1 \geq x_1, \ldots, y_n \geq x_n). \tag{9.16}$$

To simplify the calculation, we assume independence of the features. As a result, (9.16) can be expressed as a product of the individual probabilities, where

$$P(\underline{x}) = \prod_{i=1}^{n} P(y_i \geq x_i). \tag{9.17}$$

Once $P(\underline{x})$ is known, the support of $\underline{x}$ in a database of random feature vectors can be modeled as a binomial distribution. To illustrate, a random vector can be viewed as a trial and $\underline{x}$ occurring in it as "success." A database consisting m feature vectors will involve m trials for $\underline{x}$. The support of $\underline{x}$ in the database is the number of successes. Therefore, the probability of $\underline{x}$ having a support μ is

$$P(\underline{x};\mu) = \binom{m}{\mu} P(\underline{x})^{\mu}(1 - P(\underline{x}))^{m-\mu}. \tag{9.18}$$

The probability distribution function (abbr. pdf) of $\underline{x}$ can be generated from (9.18) by varying μ in the range $[0, m]$. Therefore, given an observed support μ_0 of $\underline{x}$, its p-value can be calculated by measuring the area under the pdf in the range $[\mu_0, m]$, which is

$$p\text{-}value(x, \mu_0) = \sum_{i=\mu_0}^{m} P(\underline{x}; i). \tag{9.19}$$

9.5.3.3 Identifying Regions of Interest

With the conversion of graphs into feature vectors and a model to evaluate significance of a graph region in the feature space, the next step is to explore how the feature vectors can be analyzed to extract the significant regions. Based on the feature vector representation, the presence of a "common" sub-feature vector among a set of graphs points to a common subgraph. Similarly, the absence of a "common" sub-feature vector indicates the non-existence of any common subgraph. Mathematically, the *floor* of the feature vectors produces the "common" sub-feature vector.

Definition 10 (Floor of Vectors) The floor of a set of vectors $\{\underline{v_1}, \ldots, \underline{v_m}\}$ is a vector $\underline{v_f}$ where $v_{f_i} = \min(v_{1_i}, \ldots, v_{m_i})$ for $i = 1, \ldots, n$. Ceiling of a set of vectors is defined analogously.

The next step is to mine common sub-feature vectors which are also significant. Algorithm 5 presents the FVMine algorithm which explores closed sub-vectors in a bottom-up, depth-first manner. FVMine explores all possible common vectors satisfying the significance and support constraints.

Algorithm 5 FVMine($\underline{x}$, S, b)

Input: Current sub-feature vector $\underline{x}$, supporting set S of $\underline{x}$, current starting position b
Output: The set of all significant sub-feature vectors A

1: **if** $p\text{-}value(\underline{x}) \leq maxPvalue$ **then**
2: $A \leftarrow A + x$;
3: **for** $i = b$ to m **do**
4: $S' \leftarrow \{\underline{y} | y \in S, y_i > x_i\}$;
5: **if** $|S'| < min\_sup$ **then**
6: **continue**;
7: $\underline{x'} = floor(S')$;
8: **if** $\exists j < i$ such that $x'_j > x_j$ **then**
9: **continue**;
10: **if** $p\text{-}value(ceiling(S'), |S'|) \geq maxPvalue$ **then**
11: **continue**;
12: $FVMine(\underline{x'}, S', i)$;

With a model to measure the significance of a vector, and an algorithm to mine closed significant sub-feature vectors, a new graph mining framework is built to mine significant sub-feature vectors and use them to locate similar regions that are significant. Algorithm 6 outlines the GraphSig algorithm.

Algorithm 6 GraphSig(D, min_sup, $maxPvalue$)

Input: Graph dataset D, support threshold min_sup, p-value threshold $maxPvalue$
Output: The set of all significant sub-feature vectors A

1: $D' \leftarrow \varnothing$;
2: $A \leftarrow \varnothing$;
3: **for** each $g \in D$ **do**
4: $\quad D' \leftarrow D' + RWR(g)$;
5: **for** each node label a in D **do**
6: $\quad D'_a \leftarrow \{\underline{v} | \underline{v} \in D', label(\underline{v}) = a\}$;
7: $\quad S \leftarrow FVMine(floor(D'_a), D'_a, 1)$;
8: $\quad$ **for** each vector $\underline{v} \in S$ **do**
9: $\quad\quad V \leftarrow \{u | u\ is\ a\ node\ with\ label\ a, \underline{v} \subseteq vector(u)\}$;
10: $\quad\quad E \leftarrow \varnothing$;
11: $\quad\quad$ **for** each node $u \in V$ **do**
12: $\quad\quad\quad E \leftarrow E + CutGraph(u, radius)$;
13: $\quad\quad A \leftarrow A + Maximal\_FSM(E, freq)$;

The algorithm first converts each graph into a set of feature vectors and puts all vectors together in a single set D' (lines 3–4). D' is divided into sets, such that D'_a contains all vectors produced from RWR on a node labeled a. On each set D'_a, FVMine is performed with a user-specified support and a p-value threshold to retrieve the set of significant sub-feature vectors (line 7). Given that each sub-feature vector could describe a particular subgraph, the algorithm scans the database to identify the regions where the current sub-feature vector occurs. This involves finding all nodes labeled a and described by a feature vector such that the vector is a super-vector of the current sub-feature vector $\underline{v}$ (line 9). Then the algorithm isolates the subgraph centered at each node by using a user-specified radius (line 12). This produces a set of subgraphs for each significant sub-feature vector. Next, maximal subgraph mining is performed with a high-frequency threshold since it is expected that all graphs in the set contain a common subgraph (line 13). The last step also prunes out false positives where dissimilar subgraphs are grouped into a set due to the vector representation. For the absence of a common subgraph, when frequent subgraph mining is performed on the set, no frequent subgraph will be produced and as a result the set is filtered out.

9.5.3.4 Application to Graph Classification

As a significant subgraph is able to describe a property where the data set deviates from expected, it contains more distinctive information. To utilize this potential, [25] developed a classifier built on significant patterns mined by GraphSig. The basic idea is, given a query graph, the algorithm finds the k-closest significant

subgraphs in the training set. The majority vote by the k-closest subgraphs decides the classification. The significant subgraphs in the positive or negative data set represent those subgraphs that occur more often than expected in a random database of graphs. Therefore, the occurrence of any of these subgraphs in the query acts as an identifier of its class.

Algorithm 7 outlines the pseudocode. The feature space representation of the query graph and sets of significant sub-feature vectors from the positive data set D_+ and negative data set D_- are fed to the classifier. For each node in the query graph, the distance to the closest sub-feature vector in the training set is calculated and inserted into a priority queue along with the class identifier. The priority queue keeps track of the k-closest significant sub-feature vectors for the query graph. After all nodes are scanned, a distance-weighted score is calculated for classification. The score calculation is similar to a distance-weighted k-NN classifier, where it not only considers the majority vote but also the distance between the query and its neighbor.

Algorithm 7 Classify(D_+, D_-, g, k)

Input: A set of positive significant vectors D_+, a set of negative significant vectors D_-, the vector representation of query graph g, number of neighbors k
Output: Classification of g

1: $PQ \leftarrow$ priority queue of size k;
2: $score \leftarrow 0$;
3: **for** each node $n \in g$ **do**
4: $\quad posDist \leftarrow minDist(vector(n), D_+)$;
5: $\quad negDist \leftarrow minDist(vector(n), D_-)$;
6: $\quad$ **if** $negDist < posDist$ **then**
7: $\quad\quad$ insert ($negDist$, -1) into PQ;
8: $\quad$ **else**
9: $\quad\quad$ insert ($posDist$, 1) into PQ;
10: **for** each tuple $t \in PQ$ **do**
11: $\quad score \leftarrow score + \frac{t[2]}{t[1]+\delta}$ (δ is an added small value to avoid division by zero)
12: **if** $score > 0$ **then**
13: $\quad$ **return** "positive";
14: **else**
15: $\quad$ **return** "negative";

9.5.4 Model-Based Search Tree

Fan et al. proposed $\mathsf{M}^b\mathsf{T}$ (model-based search tree) [9] for classifying graph data. It builds a decision tree that partitions the graph data onto different nodes. It starts with the whole data set and mines a set of frequent subgraphs from the data. The best pattern is selected according to some criterion and used to divide the data set into two subsets, one containing this pattern and the other not. The mining and pattern selection procedure is repeated on each of the subsets until the subset is small enough or the examples in the subset have the same class label. Since the number

of examples toward leaf level is relatively small, this approach is able to examine patterns with extremely low global support that could not be enumerated on the whole graph data set. After the algorithm completes, a small set of informative features are uncovered and the corresponding model-based search tree is constructed. The discovered feature vectors are more accurate on some graph data sets while the total feature set size is typically 50% or smaller. Importantly, the minimum support of some discriminative patterns can be extremely low (e.g., 0.03%). Algorithm 8 presents the recursive method that builds the model-based search tree.

Algorithm 8 Build model-based search tree

Input: Graph database D, support threshold min_sup, a pattern mining algorithm $fp()$, minimum node size m
Output: A selected set of features $\mathcal{F}_s$, a model-based search tree T

1: Call the frequent pattern mining algorithm which returns a set of frequent subgraphs $\mathcal{F} = fp(D, min\_sup)$;
2: Evaluate the fitness of each pattern $\alpha \in \mathcal{F}$;
3: Choose the best pattern α_m as the feature;
4: $\mathcal{F}_s = \mathcal{F}_s \cup \{\alpha_m\}$;
5: Maintain α_m as the testing feature in current node of the tree T;
6: $D_L =$ subset of examples in D containing α_m;
7: $D_R = D - D_L$;
8: **for** $\ell \in \{L, R\}$
9: **if** $|D_\ell| \leq m$ or examples in D_ℓ have the same class label
10: Make T_ℓ a leaf node;
11: **else**
12: Recursively construct T_ℓ with D_ℓ and min_sup;
13: Return $\mathcal{F}_s$ and T;

9.5.4.1 Pattern Enumeration Scalability Analysis

For a problem of size s (the data set size) and $min\_sup = p$, the recursive algorithm enumerates $O(s^{s(1-p)})$ number of subgraph patterns in total during the tree construction process. This result can be derived based on the Master Theorem as a recurrence problem.

It is worth noting that in the recursive algorithm, the support p is the support at each node, i.e., support is calculated among all records falling into the node. A subgraph pattern with support p at a node will have a global support p', which is much smaller than p. For example, assume that the leaf node size is 10 and $p = 20\%$. For a problem size of 10000, the normalized support in the complete data set is $10 \times 20\%/10000 = 0.02\%$.

To find such patterns, the traditional pattern mining algorithms will return an explosive number of patterns or fail due to resource constraints, since it will generate $O(s^{s(1-p')})$ patterns, which is a huge number. Suppose p' is close to 0, then $1-p' \simeq 1$ and the subgraph mining algorithms could obtain up to s^s patterns. However, the recursive algorithm could identify such subgraph patterns without considering

every pattern, thus will not generate an explosive number of patterns. Compared with traditional subgraph pattern mining approaches, the "scale down" ratio of the pattern numbers will be up to $\simeq s^{s(1-p)}/s^s = \frac{1}{s^{sp}}$.

9.5.4.2 Bound on Number of Returned Features

The upper bound on the number of discriminative features returned by the recursive method can be estimated as follows. In the worst case, the tree is complete and every leaf node has exactly m examples, thus the upper bound is $O(2^{\log_m(s)-1} - 1) < O(s/2 - 1) = O(s) = O(n)$, since $m > 2$ and the scaled problem size is exactly the number of examples n for the model-based search tree.

Besides theoretical analysis, experimental results demonstrate the scalability and accuracy of the $\mathsf{M}^b\mathsf{T}$ method, as the model-based search tree solves the subgraph mining problem and the graph classification problem in a divide-and-conquer way.

9.6 Conclusions

With the research achievements on frequent subgraph mining algorithms, frequent subgraph and its variations have been used for classifying structural data. Frequent subgraphs preserve the structural information of the underlying graph data; thus they are good classification features. We first introduce different types of subgraph patterns and their usage in classifying structural data. However, the inherent complexity in graph data may cause the combinatorial explosion problem in frequent subgraph mining. To avoid this problem, we further discuss several studies which directly mine the most discriminative subgraphs for classification, while avoiding the generation of many other subgraphs with low discriminative power. The subgraph pattern-based classification technique will have many real-world applications including drug design, software reliability analysis, and spam detection.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 1994 International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, Sept. 1994.
2. K. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 2005 International Conference on Data Mining (ICDM'05)*, pages 74–81, Houston, TX, Nov. 2005.
3. J. Carbonell and J. Coldstein. The Use of MMR, Diversity-Based Reranking for Reordering Documents and producing Summaries. In *Proceedings of the 1998 International Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 335–336, Melbourne, Australia, Aug. 1998.
4. H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan. Identifying bug signatures using discriminative graph mining. In *Proceedings of the 2009 International Symposium on Software Testing and Analysis (ISSTA'09)*, pages 141–151, Chicago, IL, July 2009.

5. H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 2007 International Conference on Data Engineering (ICDE'07)*, pages 716–725, Istanbul, Turkey, April 2007.
6. H. Cheng, X. Yan, J. Han, and P. S. Yu. Direct discriminative pattern mining for effective classification. In *Proceedings of the 2008 International Conference on Data Engineering (ICDE'08)*, pages 169–178, Cancun, Mexico, April 2008.
7. G. Cong, K.-Lee Tan, A. K. H. Tung, and X. Xu. Mining top-k covering rule groups for gene expression data. In *Proceedings of the 2005 International Conference on Management of Data (SIGMOD'05)*, pages 670–681, Baltimore, MD, June 2005.
8. M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transaction on Knowledge and Data Engineering*, 17:1036–1050, 2005.
9. W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu, and O. Verscheure. Direct mining of discriminative and essential graphical and itemset features via model-based search tree. In *Proceedings of the 2008 International Conference on Knowledge Discovery and Data Mining (KDD'08)*, pages 230–238, Las Vegas, NV, Aug. 2008.
10. H. Fröhlich, J. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the 2005 International Conference on Machine Learning (ICML'05)*, pages 225–232, Bonn, Germany, Aug. 2005.
11. T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th International Conference on Computational Learning Theory (COLT'03) and 7th Kernel Workshop*, page 129–143, Washington, DC, Aug. 2003.
12. T. Horväth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the 2004 International Conference on Knowledge Discovery and Data Mining (KDD'04)*, pages 158–167, Seattle, WA, Aug. 2004.
13. H. Hsu, J. A. Jones, and A. Orso. RAPID: Identifying bug signatures to support debugging activities. In *Proceedings of the 2008 International Conference on Automated Software Engineering (ASE'08)*, pages 439–442, L'Aquila, Italy, Sept. 2008.
14. H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 2003 International Conference on Machine Learning (ICML'03)*, pages 321–328, Washington, DC, Aug. 2003.
15. S. Kramer, L. Raedt, and C. Helma. Molecular feature mining in hiv data. In *Proceedings of the 2001 International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pages 136–143, San Francisco, CA, Aug. 2001.
16. T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pages 729–736, Vancouver BC Canada, Dec. 2004.
17. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 International Conference on Data Mining (ICDM'01)*, pages 313–320, San Jose, CA, Nov. 2001.
18. W. Li. Classification based on multiple association rules. In M.Sc. Thesis, School of Computing Science, Simon Fraser University, April 2001.
19. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the 1998 International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, New York, NY, Aug. 1998.
20. C. Liu, X. Yan, H. Yu, J. Han, and P. S. Yu. Mining behavior graphs for "backtrace" of noncrashing bugs. In *Proceedings of the 2005 SIAM International Conference on Data Mining (SDM'05)*, pages 286–297, Newport Beach, CA, April 2005.
21. T. M. Mitchell. *Machine Learning*. McGraw Hill, Boston, MA 1997.
22. G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.
23. J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proceedings of the 1993 European Conf. Machine Learning (ECML'93)*, pages 3–20, Vienna, Austria, 1993.

24. J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *Proceedings of the 1st Int. Workshop Mining Graphs, Trees and Sequences: In Conjunction with ECML/PKDD'03*, Cavtat-Dubrovnik, Croatia, Sept. 2003.
25. S. Ranu and A. K. Singh. GraphSig: A scalable approach to mining significant subgraphs in large graph databases. In *Proceedings of the 2009 International Conference on Data Engineering (ICDE'09)*, pages 844–855, Shanghai, China, Mar. 2009.
26. M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *Proceedings of the 2009 SIAM International Conference on Data Mining (SDM'09)*, pages 1075–1086, Sparks, NV, April 2009.
27. K. Tsuda. Entire regularization paths for graph data. In *Proceedings of the 24th International Conference on Machine Learning*, pages 919–926, 2007.
28. A. Veloso, W. Meira Jr., and M. Zaki. Lazy associative classification. In *Proceedings of the 2006 International Conference on Data Mining (ICDM'06)*, pages 645–654, Hong Kong, China, Dec. 2006.
29. N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the 2006 International Conference on Data Mining (ICDM'06)*, pages 678–689, Hong Kong, China, Dec. 2006.
30. J. Wang and G. Karypis. HARMONY: Efficiently mining the best rules for classification. In *Proceedings of the 2005 SIAM Conf. Data Mining (SDM'05)*, pages 205–216, Newport Beach, CA, April 2005.
31. X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by scalable leap search. In *Proceedings of the 2008 International Conference on Management of Data (SIGMOD'08)*, pages 433–444, Vancouver, BC, Canada, June 2008.
32. X. Yan and J. Han. CloseGraph: Mining closed frequent graph patterns. In *Proceedings of the 2003 International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pages 286–295, Washington, DC, Aug. 2003.
33. X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SDM'03)*, pages 331–335, San Francisco, CA, May 2003.

Part III
Link Analysis for Data Cleaning and Information Integration

Chapter 10
Information Integration for Graph Databases

Ee-Peng Lim, Aixin Sun, Anwitaman Datta, and Kuiyu Chang

Abstract With increasing interest in querying and analyzing graph data from multiple sources, algorithms and tools to integrate different graphs become very important. Integration of graphs can take place at the schema and instance levels. While links among graph nodes pose additional challenges to graph information integration, they can also serve as useful features for matching nodes representing real-world entities. This chapter introduces a general framework to perform graph information integration. It then gives an overview of the state-of-the-art research and tools in graph information integration.

10.1 Introduction

Graph is fast becoming an important data genre in today's database and data analysis systems and applications. Web itself is a very large graph with Web pages as nodes and links among them as edges. The Internet that makes Web possible is also a large graph with computers as nodes and network links as edges. The emergence of Web 2.0 applications further creates many other graphs that associate Web users with one another, and graphs that associate Web users with Web objects including photos (e.g., Flickr[1]), videos (e.g., Youtube[2]), and questions/answers (e.g., Yahoo! Answers[3]).

Graph is often used for data modeling or knowledge representation. Entity relationship model [6] is essentially a graph consisting of entity types and relationship types as nodes and connections among them as edges. It is widely used to design databases conceptually before the relational schemas are created. Ontology is a another kind of graph used for sharing knowledge between applications from the same domain [11]. Instead of keeping schema and data separate as in

E.-P. Lim (✉)
School of Information Systems, Singapore Management University, Singapore
e-mail: eplim@smu.edu.sg

[1] http://www.flickr.com.
[2] http://www.youtube.com
[3] http://answers.yahoo.com.

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_10,

database design, ontology uses nodes to represent both schema and data objects and edges to represent schema–level relationships (e.g., PhDStudent is a subclass of Graduate student), schema–data relationships (e.g., John and Mary are instances of PhDStudent), and data–level relationships (e.g., John is a friend of Mary). With ontology graphs defined for different knowledge domains, knowledge-based systems and applications are expected to interoperate using the same set of concepts to describe data and to perform reasoning on them. Ontology is later adopted by Semantic Web as an extension to the existing Web enabling machines to standardize the description of Web objects and services so as to understand them and deploy Web services.

Graphs also exist in relational databases although the graph structures in relational database are very much *normalized*. Records in a relational table can be associated with records from another table using foreign key references or via a relation containing many-to-many associations between record keys. For example, Internet Movie Database (IMDb) is an online relational database of movie, actor, and director-related data. These movie-related entities essentially form a graph if we view them as nodes and associations among them as edges. For example, every movie record is linked to its actors and director, and each actor is linked to all of his or her movies. Compared to ontology, graphs that are embedded in relational data are more structured and their structures are governed by the database schema.

Other than the above two schools of thought, schemaless graphs with nodes and edges assigned labels have also been widely used to represent complex structures such as protein and chemical compounds. Examples of such graph models are OEM [5, 10, 20]. With the popularity of e-commerce, such graph models have also been applied to modeling XML data so as to formulate and evaluate queries on XML databases.

Information integration [13], sometimes also known as *data integration* and *semantic integration*, refers to merging information from different data sources in order to gain a more complete set of data for developing new applications, and for conducting data analysis/data mining. The new applications to be developed can be due to the demand for new functionalities or due to application, database, or even enterprise level merger activities. Since the original databases residing at different data sources are likely administered by different parties, information integration has to address three major technical challenges arising from data heterogeneity [18, 21, 22]:

– *Schema-Level Heterogeneity*: This refers to schema differences between two or more databases to be integrated. Depending on the physical data models of the original data sources, the scheme level heterogeneity may involve matching schema elements of original databases and mapping them to the schema elements of an integrated schema [22]. In the case of schemaless graph data, schema-level heterogeneity does not exist as there is only a single node type and a single edge type.

- *Instance-Level Heterogeneity*: Data heterogeneity occurs at the instance level when data instances from different data sources but describing the same real-world entity do not look alike. The conflicts incurred can be classified into *entity identity conflicts* [16] and *attribute value conflicts* [17]. Entity identity conflicts refer to the difficulties in resolving records that model the same real-world entities due to synonym and homonym problems [16]. Attribute value conflicts refer to attribute value differences in records to be integrated together. The tasks of resolving entity identity and attribute value conflicts are known as *entity identification* (or *entity resolution*) and *attribute value conflict resolution*, respectively.
- *Federated Query Processing*: Heterogeneous databases can be integrated either physically or virtually. The former refers to migrating the original data to a central database to be physically merged and stored there. Virtual information integration refers to developing a software layer simulating an integrated database while keeping the original databases intact. The physical and virtual database integration approaches are also known as the *data warehousing* and *federated database* approaches. To evaluate queries on a federated database, one has to determine the source databases to query and to resolve data heterogeneity during query processing [15].

In the context of relational databases, the above challenges have been studied for almost three decades [24] and many interesting information integration techniques have been developed. Nevertheless, not all integration issues have be fully addressed so far as there are often hidden semantics in the data heterogeneity that are not made available to the information integration techniques that depend on them. Meanwhile, the emergence of graph data will pose additional research issues to be considered in designing information integration techniques.

This chapter will thus give an overview of information integration for graph data. Although there are applications that require different types of integrated graph data, the same integration framework and techniques can be applied. In Section 10.2, we introduce a general integration framework for graph integration before describing the techniques appropriate for the different framework components. We specifically describe a few entity resolution approaches in Section 10.3. We will highlight two graph information integration applications that have been reported in the research literature in Section 10.4. and 10.5 concludes the chapter.

10.2 Framework for Graph Information Integration

Figure 10.1 depicts the framework of integrating multiple graphs together although it is also possible to integrate two graphs at a time. The framework broadly divides the steps into *schema-level* and *instance-level integration*. The former addresses the heterogeneity of schemas of the original graphs by automatically or semi-automatically deriving the mappings between the schema elements. This is

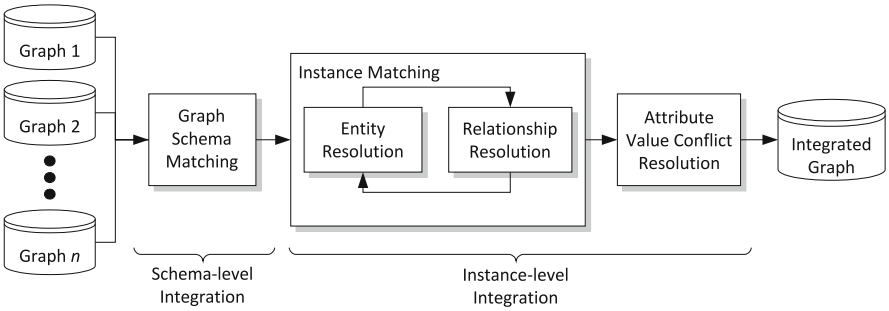


Fig. 10.1 Graph information integration framework

also known as *schema matching*. Instance-level integration refers to resolving the heterogeneity among data instances. The sub-steps include *entity resolution* and *relationship resolution* which can be carried out either sequentially or in an iterative manner. The entity resolution sub-step attempts to match graph nodes representing the same entities. The relationship resolution on the other hand matches graph edges representing the same relationships. After entity and relationship resolution, one can conduct *attribute value resolution* on the matched nodes and relationships to resolve any differences in their attributes.

The complexity involved in each integration step is correlated with the heterogeneity to be resolved. For example, in cases when the graphs to be integrated share a common schema, the graph schema matching step becomes trivial. The step can, however, be complex when (a) the underlying data models are different, (b) schemas have very different elements (i.e., node types, edge types, node attributes, and edge attributes) causing the mappings between them may not be 1-1, and (c) different constraints on the node and edge types are specified for different schemas.

Graph schema matching, depending on the data model used to represent graph instances, can be quite similar to schema matching involving relational data instances [22]. Many techniques already developed for relational schema matching are still applicable. Attribute value resolution is a step conducted on matched entities or matched relationships and is largely not affected by the graph structures. Again, the existing attribute value resolution techniques for relational data can be employed. Between entity and relationship resolutions, entity resolution has been intensively studied while relationship resolution is still a relatively less studied problem because most of the graphs to be integrated have relationships uniquely identified by their associated entities. This makes relationship resolution relatively straightforward. In other words, if entities A and B are linked with a relationship in each of two given graphs, the two relationships are identified to be the same. In this chapter, we shall therefore focus largely on the entity resolution step and its associated approaches.

Entity resolution for graph data is quite different from that for relational data due to the existence of links in graphs. When two nodes are to be determined as the same entity, one has to consider the links and even link structures connected

to the two entities. We would ideally like the two nodes to share the same links or link structures if they are the same entity. This is, however, impossible unless some of the neighboring nodes have been resolved a prior. A naive approach to break this chicken and egg situation is to ignore links and simply apply entity resolution techniques for relational data. Such an approach is non-ideal and we thus introduce other techniques that consider links.

10.3 Entity Resolution for Graphs

Entity resolution, sometimes also known as *record linkage* and *entity de-duplication*, for graphs essentially identifies nodes that model same real-world entities. When two nodes model the same real-world entity, they form a *matched entity pair*. There have been many approaches proposed to determine matched entity pairs. Some of them assume that no two nodes in the same graph model the same real-world entity. In a more general integration setting, this assumption may not hold and one may have to perform entity resolution on a single graph.

The general strategy of entity resolution is depicted by the following steps:

1. Compute similarities for entity pairs.
2. User(s) judge if the pairs are matched.

The first step heavily depends on the definition of *inter-entity similarity*. Given two or more graphs, we can measure the inter-entity similarity using a similarity function $sim(e_i, e_j)$. When each graph has N entities, the number of entity pairs for similarity computation will be N^2 in the worst case. To reduce the number of entity pairs in steps 1 and 2, a *minimum similarity threshold* can be introduced. Omar et al., in the context of relational database integration exploited properties of the matching and merging entities to reduce the computation and judgment overheads of entity resolution [1].

Bhattacharya and Getoor provided a comprehensive survey of entity resolution approaches in [3]. In the following, we adopt their classification of entity resolution approaches and briefly describe them. Entity resolution can be solved in both supervised or non-supervised approaches. The former requires training data while the latter does not. In the research literature, there is a variety of supervised entity resolution approaches [8, 23] but due to space constraint, we will only cover the non-supervised ones below.

10.3.1 Attribute-Based Entity Resolution

Attribute-based entity resolution compares the attribute values of entities in order to match them. It has been widely used in resolving entity identities in relational databases and graphs due to its simplicity. Attributes or combinations of attributes that can identify entities either strongly or weakly are used in attribute-based entity

resolution approaches. Attributes or attribute combinations that strongly identify entities are the entity identifiers, e.g., social security number (or SSN), person name and birthdate, and mobile number. The other attributes or attribute combinations that increase the odds of identifying entities are then known to identify entities weakly, e.g., birth year, land phone number, home address, and company address. According to the nature of the attributes used in entity resolution, there can be a few different ways of defining entity level similarity function $sim(e_i, e_j)$.

The $sim_{attrib}(e_i, e_j)$ can be defined to be either 0 or 1 only depending on the outcome of comparing the entity attributes $e_i.attrib$ and $e_j.attrib$. For example, the following inter-entity similarity function $sim_{\text{SSN}}(e_i, e_j)$ returns a binary value whenever e_i and e_j have identical social security number. The second function $sim_{\text{homeaddress,birthyear}}(e_i, e_j)$ returns *Jaccard* similarity metric value of their home address, a value between 0 and 1, indicating how likely e_i and e_j are matched entities when their birth year values are the same, and 0 when the birth year values are different:

$$sim_{\text{SSN}}(e_i, e_j) = \begin{cases} 1 & \text{if } e_i.\text{SSN} = e_j.\text{SSN} \\ 0 & \text{otherwise,} \end{cases}$$

$$sim_{\text{home address,birth year}}(e_i, e_j) = \begin{cases} Jaccard(e_i.\text{home address}, & \text{if } e_i.\text{birth year} = \\ \qquad e_j.\text{home address}) & \quad e_j.\text{birth year} \\ 0 & \text{otherwise.} \end{cases}$$

The *Jaccard* similarity metric of two strings of word tokens s_i and s_j is defined by

$$Jaccard(s_i, s_j) = \frac{\text{number of common words between } s_i \text{ and } s_j}{\text{number of distinct words in } s_i \text{ and } s_j}.$$

A survey of similarity metrics for string attributes such as *Jaccard* can be found in [7]. Similarity metrics for numeric attributes include normalized difference, cosine similarity, and euclidean distance [14]. Given that entities usually have multiple attributes, these similarity metrics can be combined in various ways to determine entity similarity.

10.3.2 Relational Entity Resolution

Relational entity resolution essentially involves the use of link connectivity of entities to determine how similar the entities are. Bhattacharya and Getoor proposed several relational entity resolution approaches [3] to improve over using direct attribute entity resolution.

In the *naïve relational entity resolution* approach [3], the inter-entity similarity is derived by applying an attribute-based similarity function $sim_A(e_i, e_j)$ and another edge-based similarity $sim_H(e_i, e_j)$ on the pair of entities to be matched. The overall inter-entity similarity is defined by

$$sim(e_i, e_j) = (1 - \alpha) \cdot sim_A(e_i, e_j) + \alpha \cdot sim_H(e_i, e_j),$$

where $0 \leq \alpha \leq 1$. The function $sim_H(e_i, e_j)$ is determined by the similarity of edges of e_i and e_j and can be defined as the aggregated similarity between the edges of e_i and e_j (denoted by $e_i.edges$ and $e_j.edges$ respectively), i.e.,

$$sim_H(e_i, e_j) = \sum_{l_i \in e_i.edges, l_j \in e_j.edges} sim_H(l_i, l_j).$$

The similarity between a pair of edges can then be defined by

$$sim_H(l_i, l_j) = Max_{e_s \in E_i, e_t \in E_j} sim_A(e_s, e_t),$$

where E_i and E_j denote the entities linked to l_i and l_j, respectively.

In the *Simrank* approach proposed by Jeh and Widom [12], the inter-entity similarity is defined by a random walk process on neighbors of entity pairs. Suppose a graph is directed and I_i (and I_j) denotes the set of in-neighbors of e_i (and e_j), the Simrank similarity between entities e_i and e_j is defined by

$$sim_{simrank}(e_i, e_j) = \begin{cases} 1 & \text{if } e_i = e_j \\ \frac{C}{|I_i||I_j|} \sum_{e'_i \in I_i} \sum_{e'_i \in I_i} sim_{simrank}\left(e'_i, e'_j\right) & \text{otherwise,} \end{cases}$$

where C is a decay factor constant between 0 and 1. Unlike the earlier approach, Simrank does not use attribute-based similarity function at all. Simrank also requires the graph to be directed. For it to work on undirected graph, a simple way is to replace an undirected edge by two directed edges, one for each direction.

10.3.3 Collective Relational Entity Resolution

The main idea of collective relational entity resolution is to group entities into *entity clusters* each representing a group of entities that model a real-world entity. The inter-entity similarity is thus measured by a combination of how similar a pair of entities e_i and e_j are by their attributes, and how similar they are by the cluster labels of their neighbors. Bhattacharya and Getoor proposed an agglomerative clustering algorithm that group entity clusters into larger entity clusters incrementally using the inter-entity cluster similarity function [3]:

$$sim(c_i, c_j) = (1 - \alpha) \cdot sim_A(c_i, c_j) + \alpha \cdot sim_R(c_i, c_j),$$

where $sim_A(c_i, c_j)$ and $sim_R(c_i, c_j)$ represent the attribute-based and relational similarities between two entity clusters c_i and c_j, respectively. The former can be determined by an aggregated attribute-based similarity between entities from c_i and c_j. The latter, $sim_R(c_i, c_j)$, is determined by the number of common cluster labels among the neighbors of entities in c_i and c_j. This neighborhood similarity can be measured using a variety of functions including *common neighbors*, *jaccard similarity*.

Instead of agglomerative clustering, Bhattacharya and Getoor also introduced a Latent Dirichlet Allocation (LDA) model for conducting collective relational entity resolution for authors of a set of publications using probabilistic model [2]. Here, author entity clusters are represented by latent authors. The observed author entities are assumed to be generated by these latent authors. Learning the mapping from observed author entities to latent authors is thus a problem of learning LDA model parameters.

10.4 Example Applications

In this section, we shall describe two example applications of entity resolution to social network analysis. The first is *D-Dupe*, an interactive tool for entity resolution [4]. The second is *SSnetViz*, an application to visualize and explore integrated heterogeneous social networks [19].

10.4.1 D-Dupe

D-Dupe is an interactive application specially designed to resolve entity identities for authors of publications [4]. The graphs to be integrated have authors as nodes and co-authorships as edges. Each edge is thus associated with a set of publications between the connected authors. Figure 10.2 shows the user interface design of D-Dupe using the Infoviz data set that comes with the application demo [9]. It consists of mainly three user interface components including

- *Search Panel*: The search panel shows a list of entity pairs ranked by their inter-entity similarity values. Different attribute and relational similarity metrics can be chosen by the user. The panel also supports string search on author entities so as to find potential duplicates of the selected authors.
- *Graph Visualizer*: The graph visualizer displays a subgraph with a selected candidate pair of duplicate authors (e.g., "Lucy Nowell" and "Lucy T. Nowell" in the figure), their common co-authors in-between ("Deborah Hix" in the figure) and other non-common co-authors at the sides. The purpose of graph visualizer is to allow user to easily tell whether the candidate duplicate authors are indeed the same author. The user can then decide whether to merge the two candidate authors and to mark them distinct.

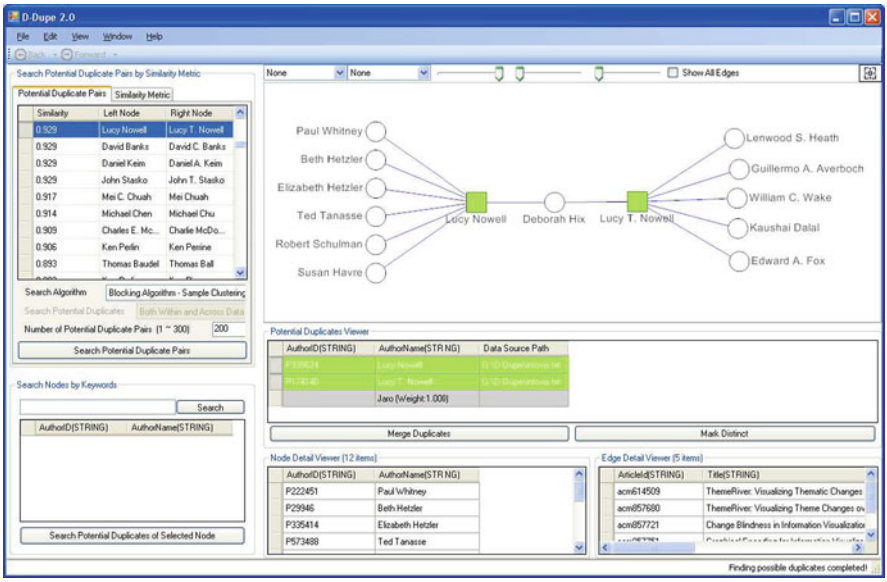


Fig. 10.2 User interface of D-Dupe

– *Detail Viewer*: The detail viewer provides detailed views of selected author candidates, their co-authors' nodes and edges in the graph visualizer.

The layout of the author graph is such that there is always some node(s) between the selected author candidates as it is not possible to have both of them appearing in the same publication. The size of an author node can be based on the author's number of publications. Both nodes and edges can be filtered for clearer viewing. Once the user selects two candidate authors to be merged together, the graph visualizer will show the two nodes combined into one and their two sets of edges are assigned to the new node labeled by "Lucy T. Nowell" as shown in Fig. 10.3. D-Dupe automatically selects the next node most similar to the merged node, i.e., "Lucy Terry Nowell," as a candidate entity to be merged. The entity resolution steps can repeat till all author entities are appropriately merged.

10.4.2 SSnetViz

SSnetViz, unlike D-Dupe, is an ongoing research project to explore and analyze heterogeneous social networks integrated from multiple data sources [19]. A heterogeneous social network is one with multiple node types (or entity types) and link types (or relationship types). In SSnetViz, nodes of the same type share the same attribute set while links are not assigned any attributes. SSnetViz provides some functions to integrate multiple heterogeneous social networks and data exploration features that allow the data sources to be identified even after entities are merged.

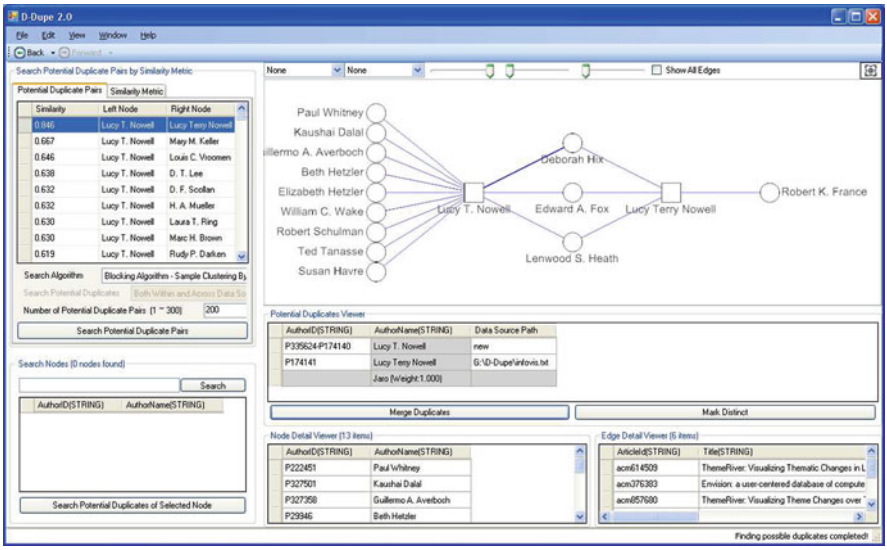


Fig. 10.3 After merging "Lucy Nowell" and "Lucy T. Nowell"

The design of SSnetViz is originally motivated by the need to support data analysis of terrorism related social networks from disparate sources. The networks SSnetViz has so far integrated come from three data sources: (a) TKBNetwork: a terrorism network from the MIPT Terrorism Knowledge Base[4] (TKB); (b) PVTRNetwork: a terrorism network created by the International Center for Political Violence and Terrorism Research (ICPVTR), a center focusing on gathering and analysis terrorism data using public domain data; and (c) WikiTerrorism: a network we constructed using terrorism-related articles from Wikipedia. Both TKBNetwork and PVTRNetwork are constructed by experts while WikiTerrorism represents knowledge collaboratively edited by the online community.

Figure 10.4 depicts the user interface design of SSnetViz. It consists of a *network viewer* that displays the entire social network graph or selected subgraph. Nodes of different types are shown using different shapes while the different data sources are shown using different colors. The legend of shapes and colors of nodes and links is shown at the bottom of network viewer. For example, terrorists are shown as oval nodes and terrorist groups are shown as rectangle nodes. Light blue and yellow indicate the information from TKBNetwork and PVTRNetwork respectively. When a node or link is derived by integrating two or more original networks (e.g., "Jemaah Islamiya (JI)" node in Fig. 10.4), we assign them a distinct color representing the overlapping data sources. The color and shape schemes can be configured by users for easy viewing. Zooming, rotation, hyperbolic view, node expansion, node/link

[4] MIPT is the acronym of Memorial Institute for the Prevention of Terrorism.

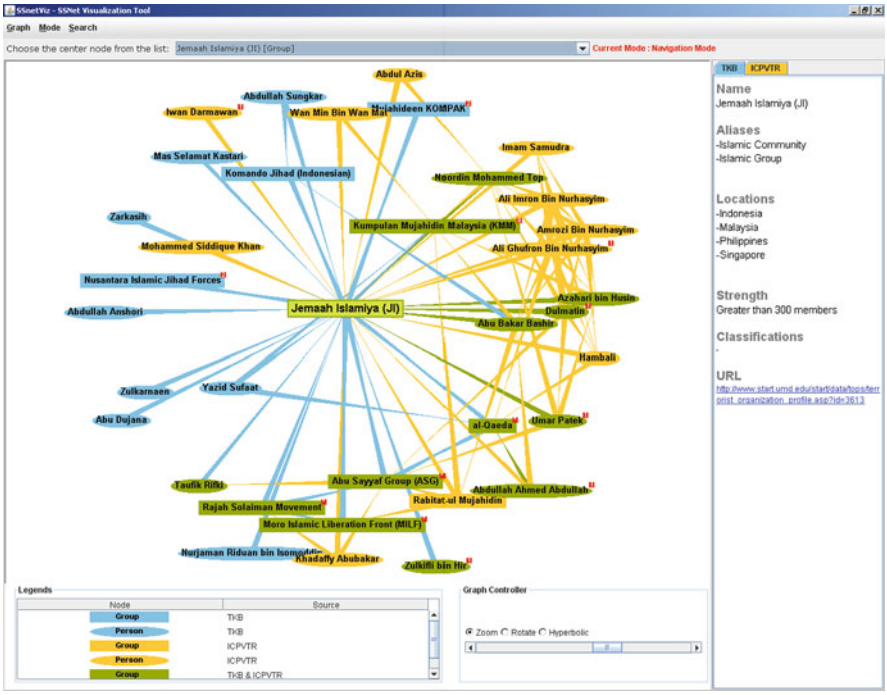


Fig. 10.4 User interface design of SSnetViz

hiding, node search, path search, and other operations are also provided to manipulate and explore the social network in network viewer. Whenever a node is selected, its attribute values divided according to the data source(s) the node is found will be shown in the node information panel on the right side. In Fig. 10.4, "Jemaah Islamiya (JI)" node's attribute values from TKBNetwork and PVTRNetwork are shown.

Since SSnetViz has to integrate social networks with heterogeneous schemas and node/link instances, both schema-level integration and instance-level integration are required. The integration steps depicted in Fig. 10.5 have been adopted. SSnetViz has a generic graph schema that is capable of storing and updating social networks with heterogeneous schemas. When two social networks are to be integrated, the user first matches the node types of the networks using a *node type matching module*. The step is performed once only and the matched node types are stored for subsequent instance-level integration.

Instance-level integration in SSnetViz involves mainly node matching, i.e., entity resolution. Link instances are automatically integrated if their link types are identical. Both rule based and manual node matching approaches are supported by a *node merging module* shown in Fig. 10.8.

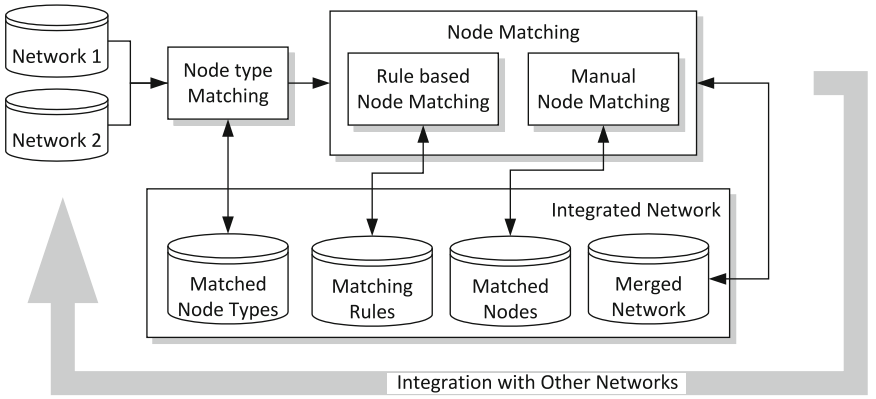


Fig. 10.5 SSnetViz's integration steps

For illustration, we show two social network subgraphs showing the one-hop neighborhoods of "Jemaah Islamiyah (JI)" from TKBNetwork and PVTRNetwork in Figs. 10.6 and 10.7 respectively. Note that the TKBNetwork and PVTRNetwork have several discrepancies in naming their entities. We would like to match their

Fig. 10.6 Social network subgraph from TKBNetwork

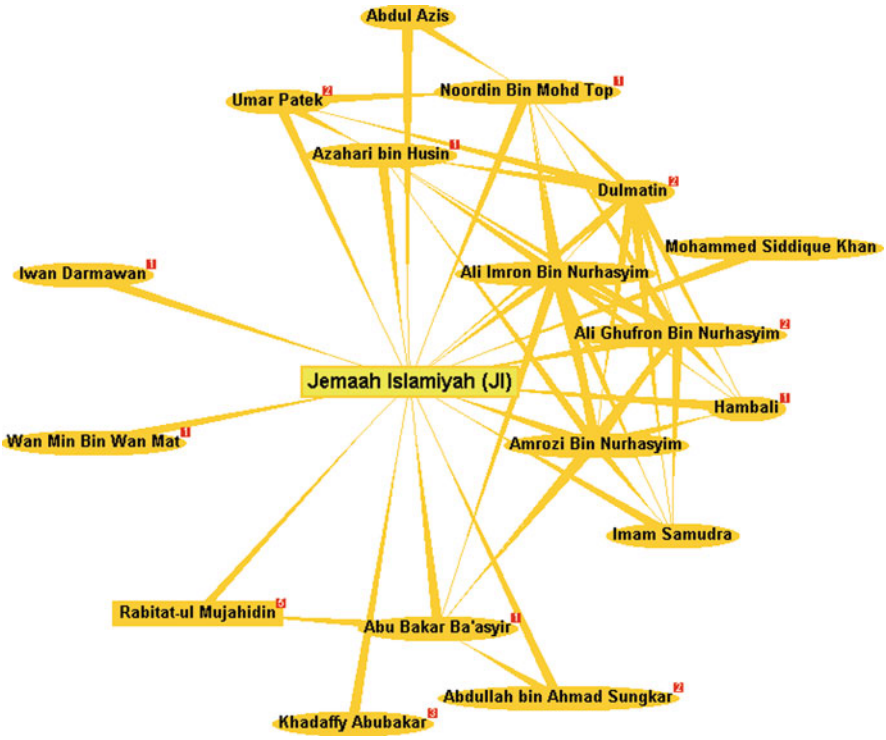


Fig. 10.7 Social network subgraph from PVTRNetwork

nodes and derive the integrated network. Table 10.1 shows a subset of matched nodes that need to be identified. Note that some of these entities have identical names in TKBNetwork and PVTRNetwork but others have some name differences, e.g., "Jemaah Islamiya (JI)" and "Jemaah Islamiyah (JI)."

In general, node instances modeling the same real-world entities may have different attribute values. SSnetViz users can match them using user-defined matching rules which specify the attribute similarity functions for generating candidate matched node pairs (see Fig. 10.8). Among the candidate matched node pairs, the user can manually merge the correct node pairs and mark the rest as incorrect ("no"

Table 10.1 Matched node pairs

	TKBNetwork	PVTRNetwork
1	Jemaah Islamiya (JI)	Jemaah Islamiyah (JI)
2	Umar Patek	Umar Patek
3	Noordin Mohammed Top	Noordin Bin Mohd Top
4	Azahari bin Husin	Azahari bin Husin
5	Dulmatin	Dulmatin
6	Abu Bakar Bashir	Abu Bakar Ba'asyir

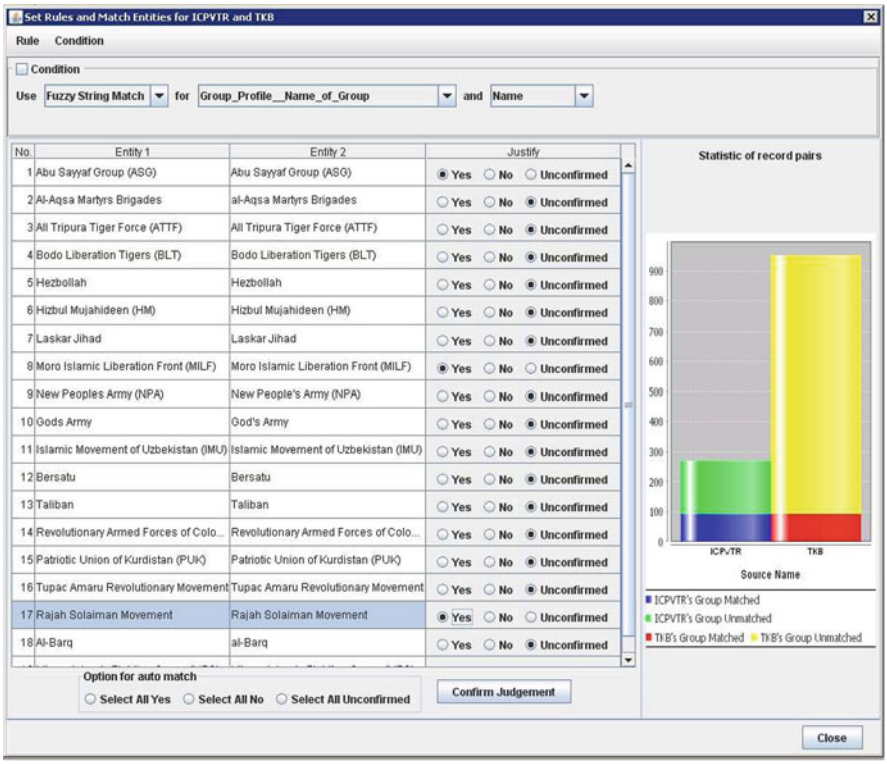


Fig. 10.8 SSnetViz's node merging module

option in the figure) or unconfirmed. A cylindrical bar showing the numbers of matched and unmatched node instances in each social network is also shown by the node merging module. In cases where matching rules fail to include a matched node pair, a manual node merging module can be used where matched node pairs can be identified manually (see Fig. 10.9). Using these two matching approaches, the integrated network can be derived. Fig. 10.10 shows the one-hop neighborhood of "Jemaah Islamiyah (JI)" in the integrated network.

10.5 Conclusion

Graph information integration is an important class of data integration problem as graph data can be found in many databases and integrated graphs are extremely useful for complex queries, data analysis, and data mining. This chapter aims to give an overview of the integration framework. We focus on entity resolution in

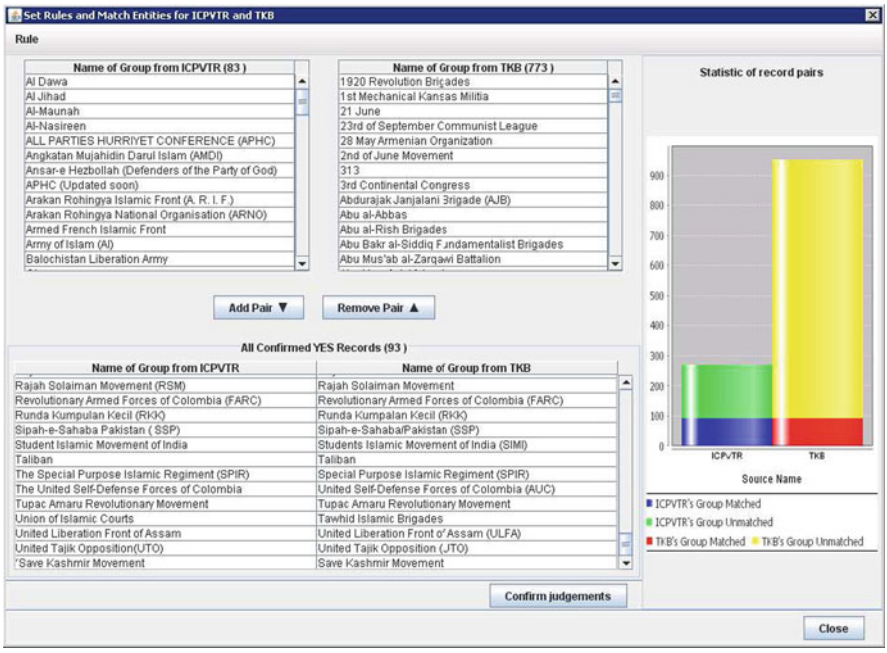


Fig. 10.9 SSnetViz's manual node merging module

graphs which has made significant progress in recent years. We also introduce two example applications, D-Dupe that is specifically designed for performing interactive entity resolution on graphs, and SSnetViz that integrates and manages integrated heterogeneous social networks.

Looking ahead, there are still many interesting problems to be addressed in graph information integration. Most of the research today has largely focused on specific problems, e.g., entity resolution, without considering the other related integration problems, e.g., schema integration and attribute value conflict resolution. Designing a complete suite of solutions for all the integration problems is challenging but is certainly the direction to pursue as the cost of data integration will only increase with more graph data to be generated in the future.

At present, researchers in graph information integration are also struggling with the evaluation of different graph information integration methods. This is mainly caused by a lack of publicly available graph data sets and methodology for performance comparison. DBLP is a good example of graph data that is publicly available. The ground truths of its integration with other author–paper graphs are, however, not available. Hence, it is difficult to compare accuracies of different integration methods on the data set.

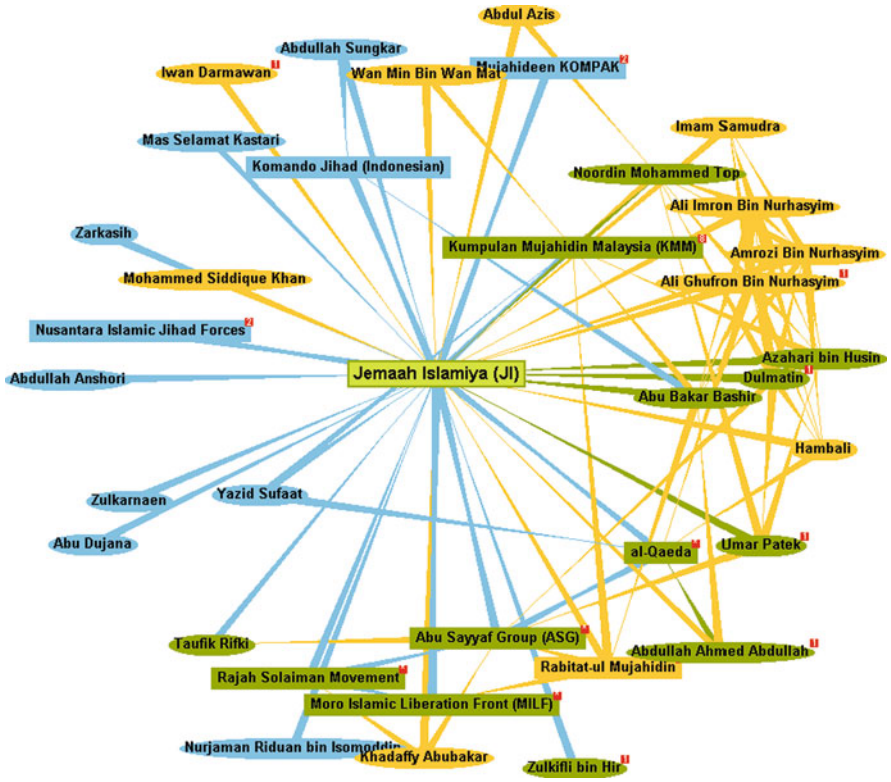


Fig. 10.10 Integrated social network subgraph

Acknowledgments We would like to acknowledge the support by A*STAR Public Sector R&D, Singapore, Project Number 062 101 0031 in the SSNet Project. We also thank Maureen and Nelman Lubis Ibrahim for implementing the SSnetViz system.

References

1. O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: A generic approach to entity resolution. *VLDB Journal*, 18(1):255–276, 2009.
2. I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *SIAM Conference on Data Mining*, Bethesda, Maryland, USA, 2006.
3. I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.
4. M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. In *International Symposium on Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 505–507, September 2005.
5. P. Buneman. Semistructured data. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, 1997.
6. P. Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

7. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IJCAI Workshop on Information Integration*, pages 73–78, Acapulco, Mexico, August 2003.
8. P. Domingos. Multi-relational record linkage. In *KDD-2004 Workshop on Multi-Relational Data Mining*, pages 31–48, Seattle, Washington, 2004.
9. J.-D. Fekete, G. Grinstein, and C. Plaisant. The history of infovis. In *IEEE InfoVis 2004 Contest*, www.cs.umd.edu/hcil/iv04contest, Austin, Texas, 2004.
10. M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Strudel: a web site management system. In *ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, 1997.
11. N. Guarino. *Formal Ontology in Information Systems*, chapter Formal Ontology in Information Systems. IOS Press, Amsterdam, 1998.
12. G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543, Edmonton, Alberta, Canada, 2002.
13. A. Jhingran, N. Mattos, and H. Pirahesh. Information integration: A research agenda. *IBM Systems Journal*, 41(4):555–562, 2002.
14. L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *International Conference on Database Systems for Advanced Applications*, Kyoto, Japan, 2003.
15. E.-P. Lim and J. Srivastava. Query optimization and processing in federated database systems. In *ACM Conference on Information and Knowledge Management*, pages 720–722, Washington D.C., 1993.
16. E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. In *IEEE International Conference on Data Engineering*, pages 294–301, Vienna, Austria, 1993.
17. E.-P. Lim, J. Srivastava, and S. Shekhar. An evidential reasoning approach to attribute value conflict resolution in database integration. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):707–723, 1996.
18. W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Survey*, 22(3):267–293, 1990.
19. Maureen, A. Sun, E.-P. Lim, A. Datta, and K. Chang. On visualizing heterogeneous semantic networks from multiple data sources. In *International Conference on Asian Digital Libraries*, pages 266–275, Bali, Indonesia, 2008.
20. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3), 1997.
21. A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Survey*, 22(3):183–236, 1990.
22. S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, 1994.
23. P. Treeratpituk and C. L. Giles. Disambiguating authors in academic publications using random forests. In *Joint Conference in Digital Libraries*, Austin, Texas, June 2009.
24. P. Ziegler and K. R. Dittrich. Three decades of data integration — all problems solved? In *18th IFIP World Computer Congress (WCC 2004)*, pages 3–12, Toulouse, France, 2004.

Chapter 11
Veracity Analysis and Object Distinction

Xiaoxin Yin, Jiawei Han, and Philip S. Yu

Abstract The World Wide Web has become the most important information source for most of us. Unfortunately, there is no guarantee for the correctness of information on the web, and different web sites often provide conflicting information on a subject. In this section we study two problems about correctness of information on the web. The first one is *Veracity*, i.e., *conformity to truth*, which studies how to find true facts from a large amount of conflicting information on many subjects that is provided by various web sites. We design a general framework for the Veracity problem, and invent an algorithm called TRUTHFINDER, which utilizes the relationships between web sites and their information, i.e., *a web site is trustworthy if it provides many pieces of true information, and a piece of information is likely to be true if it is provided by many trustworthy web sites*. The second problem is *object distinction*, i.e., how to distinguish different people or objects sharing identical names. This is a nontrivial task, especially when only very limited information is associated with each person or object. We develop a general object distinction methodology called DISTINCT, which combines two complementary measures for relational similarity: *set resemblance of neighbor tuples* and *random walk probability*, and analyze subtle linkages effectively. The method takes a set of distinguishable objects in the database as training set without seeking for manually labeled data and applies SVM to weigh different types of linkages.

11.1 Overview

The World Wide Web has become a necessary part of our lives, and might have become the most important information source for most people. Everyday people retrieve all kinds of information from the web. For example, when shopping online, people find product specifications from web sites like Amazon.com or ShopZilla.com. When looking for interesting DVDs, they get information and read

X. Yin (✉)
Microsoft Research, Redmond, WA 98052, USA
e-mail: xyin@microsoft.com

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_11, © Springer Science+Business Media, LLC 2010

movie reviews on web sites such as NetFlix.com or IMDB.com. When searching for publications of researchers, they go to DBLP or ACM Portal.

However, the World Wide Web does not always provide accurate information. There is no guarantee for the completeness or correctness of information on the web, and different web sites often provide conflicting information.

Example 1 (Authors of books) We tried to find out who wrote the book "Rapid Contextual Design" (ISBN: 0123540518). We found many different sets of authors from different online bookstores, and we show several of them in Table 11.1. From the image of the book cover we found that *A1 Books* provides the most accurate information. In comparison, the information from *Powell's books* is incomplete, and that from *Lakeside books* is incorrect.

Table 11.1 Conflicting information about book authors

Web site	Authors
A1 Books	Karen Holtzblatt, Jessamyn Burns Wendell, Shelley Wood
Powell's books	Holtzblatt, Karen
Cornwall books	Holtzblatt-Karen, Wendell-Jessamyn Burns, Wood
Mellon's books	Wendell, Jessamyn
Lakeside books	Wendell, Jessamynholtzblatt, Karenwood, Shelley
Blackwell online	Wendell, Jessamyn, Holtzblatt, Karen, Wood, Shelley
Barnes & Noble	Karen Holtzblatt, Jessamyn Wendell, Shelley Wood

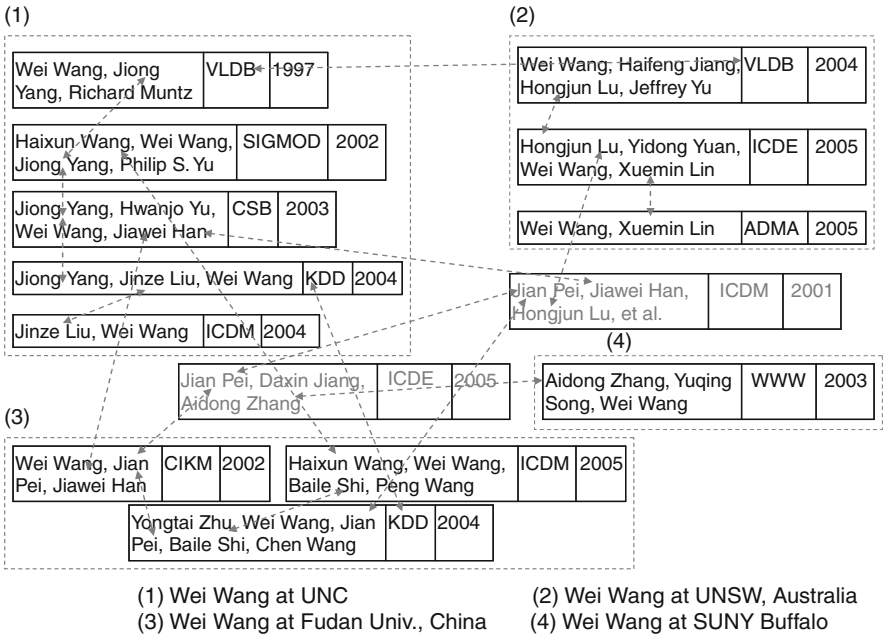


Fig. 11.1 Papers by four different "Wei Wang's"

Example 2 (People or objects with identical names) There are more than 200 papers in DBLP written by at least 14 different "Wei Wang"s, each having at least two papers. A mini example is shown in Fig. 11.1, which contains some papers by four different "Wei Wang"s and the linkages among them. Users are often unable to distinguish them, because the same person or object may appear in very different contexts, and there is often limited and noisy information associated with each appearance.

In this chapter we studied two problems about the veracity of information on the web and trustworthiness of web sites. The first problem is that, given the conflicting information from many information providers (e.g., web sites), how to find trustable information providers and accurate information. The second problem is about how to distinguish different people or objects with identical names.

11.2 Veracity Analysis of Information with Multiple Conflicting Information Providers on the Web

The trustworthiness problem of the web has been realized by today's Internet users. According to a survey on credibility of web sites conducted by Princeton Survey Research in 2005 [1], 54% of Internet users trust news web sites at least most of time, while this ratio is only 26% for web sites that sell products and is merely 12% for blogs.

There have been many studies on ranking web pages according to authority based on hyperlinks, such as Authority-Hub analysis [2], PageRank [3], and more general link-based analysis [4]. But does authority or popularity of web sites lead to accuracy of information? The answer is unfortunately no. For example, according to our experiments the bookstores ranked on top by Google (*Barnes & Noble* and *Powell's books*) contain many errors on book author information, and many small bookstores (e.g., *A1 Books*, *TheSaintBookstore*) provide more accurate information. Another example is that some large movie web sites (e.g., *www.movieweb.com*) provide less accurate information on movie runtime than some smaller movie web sites (e.g., *dvddb.sparkyb.net*) [5].

We propose a problem called *Veracity* problem, which is formulated as follows: Given a large amount of conflicting information about many objects, which is provided by multiple web sites (or other types of information providers), how to discover the true fact about each object. We use the word "fact" to represent something that is claimed as a fact by some web site, and such a fact can be either true or false. There are often conflicting facts on the web, such as different sets of authors for a book. There are also many web sites, some of which are more trustworthy than some others. A fact is likely to be true if it is provided by trustworthy web sites (especially if by many of them). A web site is trustworthy if most facts it provides are true.

Because of this interdependency between facts and web sites, we choose an iterative computational method. At each iteration, the probabilities of facts being

true and the trustworthiness of web sites are inferred from each other. This iterative procedure shares some similarity with the Authority-Hub analysis [2], but is also different in two ways. The first difference is in the definitions. The trustworthiness of a web site does not depend on how many facts it provides, but on the accuracy of those facts. Nor can we compute the probability of a fact being true by adding up the trustworthiness of web sites providing it. These lead to non-linearity in computation. Second and more importantly, different facts influence each other. For example, if a web site says a book is written by "Jessamyn Wendell," and another says "Jessamyn Burns Wendell," then these two web sites actually support each other although they provide slightly different facts.

11.2.1 Problem Definitions

The input of TRUTHFINDER is a large number of facts about properties of a certain type of objects. The facts are provided by many web sites. There are usually multiple conflicting facts from different web sites for each object, and the goal of TRUTHFINDER is to identify the true fact among them. Figure 11.2 shows a mini example data set. Each web site provides at most one fact for an object.

We first introduce the two most important definitions in this chapter, the confidence of facts and the trustworthiness of web sites.

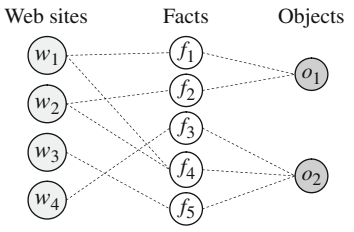


Fig. 11.2 Input of TRUTHFINDER

Definition 1 (Confidence of facts) The confidence of a fact f (denoted by $s(f)$) is the probability of f being correct, according to the best of our knowledge.

Definition 2 (Trustworthiness of web sites) The trustworthiness of a web site w (denoted by $t(w)$) is the expected confidence of the facts provided by w.

Different facts about the same object may be conflicting. However, sometimes facts may be supportive to each other although they are slightly different. For example, one web site claims the author to be "Jennifer Widom" and another one claims "J. Widom." If one of them is true, the other is also likely to be true. In this chapter we only study time-invariant facts.

In order to represent such relationships, we propose the concept of ***implication*** *between facts*. The implication from fact f_1 to f_2, $imp(f_1 \rightarrow f_2)$, is f_1's influence on f_2's confidence, i.e., how much f_2's confidence should be increased

(or decreased) according to f_1's confidence. It is required that $imp(f_1 \rightarrow f_2)$ is a value between -1 and 1. A positive value indicates if f_1 is correct, f_2 is likely to be correct. While a negative value means if f_1 is correct, f_2 is likely to be wrong. The details about this will be described in Section 11.2.2.2.

Please notice that the definition of implication is domain specific. When a user uses TRUTHFINDER on a certain domain, he should provide the definition of implication between facts. If in a domain the relationship between two facts is symmetric, and the definition of similarity is available, the user can define $imp(f_1 \rightarrow f_2) = sim(f_1, f_2) - base\_sim$, where $sim(f_1, f_2)$ is the similarity between f_1 and f_2, and $base\_sim$ is a threshold for similarity.

Based on common sense and our observations on real data, we have four basic heuristics that serve as the bases of our computational model.

Heuristic 1: *Usually there is only one true fact for a property of an object.*
Heuristic 2: *This true fact appears to be the same or similar on different web sites.*
Heuristic 3: *The false facts on different web sites are less likely to be the same or similar.*
Heuristic 4: *In a certain domain, a web site that provides mostly true facts for many objects will likely provide true facts for other objects.*

11.2.2 Computational Model

Based on the above heuristics, we know if a fact is provided by many trustworthy web sites, it is likely to be true; if a fact is conflicting with the facts provided by many trustworthy web sites, it is unlikely to be true. On the other hand, a web site is trustworthy if it provides facts with high confidence. We can see that the web site trustworthiness and fact confidence are determined by each other, and we can use an iterative method to compute both. Because true facts are more consistent than false facts (Heuristic 3), it is likely that we can distinguish true facts from false ones at the end.

In this section we discuss the computational model. Section 11.2.2.1 introduces how to infer trustworthiness of web sites and confidence of facts from each other. Section 11.2.2.2 describes how the confidence of related facts affect each other. Section 11.2.2.3 explains how we handle some additional subtlety.

11.2.2.1 Web Site Trustworthiness and Fact Confidence

We first discuss how to infer web site trustworthiness and fact confidence from each other.

As defined in Definition 2, the trustworthiness of a web site is just the expected confidence of facts it provides. For web site w, we compute its trustworthiness $t(w)$ by calculating the average confidence of facts provided by w:

Table 11.2 Variables and Parameters of TRUTHFINDER

Name	Description
M	Number of web sites
N	Number of facts
w	A web site
$t(w)$	The trustworthiness of w
$\tau(w)$	The trustworthiness score of w
$F(w)$	The set of facts provided by w
f	A fact
$s(f)$	The confidence of f
$\sigma(f)$	The confidence score of f
$\sigma^*(f)$	The adjusted confidence score of f
$W(f)$	The set of web sites providing f
$o(f)$	The object that f is about
$imp(f_j \rightarrow f_k)$	Implication from f_j to f_k
ρ	Weight of objects about the same object
γ	Dampening factor
δ	Max difference between two iterations

$$t(w) = \frac{\sum_{f \in F(w)} s(f)}{|F(w)|}, \tag{11.1}$$

where $F(w)$ is the set of facts provided by w.

In comparison, it is much more difficult to estimate the confidence of a fact. As shown in Fig. 11.3, the confidence of a fact f_1 is determined by the web sites providing it, and other facts about the same object.

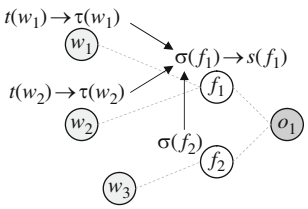


Fig. 11.3 Computing confidence of a fact

Let us first analyze the simple case where there is no related fact, and f_1 is the only fact about object o_1 (i.e., f_2 does not exist in Fig. 11.3). Because f_1 is provided by w_1 and w_2, if f_1 is wrong, then both w_1 and w_2 are wrong. We first assume w_1 and w_2 are independent. (This is not true in many cases and we will compensate for it later.) Thus the probability that both of them are wrong is $(1 - t(w_1)) \cdot (1 - t(w_2))$, and the probability that f_1 is not wrong is $1 - (1 - t(w_1)) \cdot (1 - t(w_2))$. In general, if a fact f is the only fact about an object, then its confidence $s(f)$ can be computed as

$$s(f) = 1 - \prod_{w \in W(f)} (1 - t(w)), \tag{11.2}$$

where $W(f)$ is the set of web sites providing f.

In (11.2), $1 - t(w)$ is usually quite small and multiplying many of them may lead to underflow. In order to facilitate computation and veracity exploration, we use logarithm and define the *trustworthiness score* of a web site as

$$\tau(w) = -\ln(1 - t(w)). \tag{11.3}$$

$\tau(w)$ is between and 0 and $+\infty$, which better characterizes how accurate w is. For example, suppose there are two web sites w_1 and w_2 with trustworthiness $t(w_1) = 0.9$ and $t(w_2) = 0.99$. We can see that w_2 is much more accurate than w_1, but their trustworthiness do not differ much as $t(w_2) = 1.1 \times t(w_1)$. If we measure their accuracy with trustworthiness score, we will find $\tau(w_2) = 2 \times \tau(w_1)$, which better represents the accuracy of web sites.

Similarly, we define the *confidence score* of a fact as

$$\sigma(f) = -\ln(1 - s(f)). \tag{11.4}$$

A very useful property is that the confidence score of a fact f is just the sum of the trustworthiness scores of web sites providing f. This is shown in the following lemma.

Lemma 1

$$\sigma(f) = \sum_{w \in W(f)} \tau(w). \tag{11.5}$$

Proof According to (11.2),

$$1 - s(f) = \prod_{w \in W(f)} (1 - t(w)).$$

Take logarithm on both sides and we have

$$\begin{aligned} & \ln(1 - s(f)) = \textstyle\sum_{w \in W(f)} \ln(1 - t(w)), \\ \Longleftrightarrow \quad & \sigma(f) = \textstyle\sum_{w \in W(f)} \tau(w). \end{aligned}$$

11.2.2.2 Influences Between Facts

The above discussion shows how to compute the confidence of a fact that is the only fact about an object. However, there are usually many different facts about an object (such as f_1 and f_2 in Fig. 11.3), and these facts influence each other. Suppose in Fig. 11.3 the implication from f_2 to f_1 is very high (e.g., they are very similar). If f_2 is provided by many trustworthy web sites, then f_1 is also somehow supported by these web sites, and f_1 should have reasonably high confidence. Therefore, we should increase the confidence score of f_1 according to the confidence score of f_2,

which is the sum of trustworthiness scores of web sites providing f_2. We define the *adjusted confidence score* of a fact f as

$$\sigma^*(f) = \sigma(f) + \rho \cdot \sum_{o(f')=o(f)} \sigma(f') \cdot imp(f' \rightarrow f). \tag{11.6}$$

ρ is a parameter between 0 and 1, which controls the influence of related facts. We can see that $\sigma^*(f)$ is the sum of confidence score of f and a portion of the confidence score of each related fact f' multiplies the implication from f' to f. Please notice that $imp(f' \rightarrow f) < 0$ when f is conflicting with f'.

We can compute the confidence of f based on $\sigma^*(f)$ in the same way as computing it based on $\sigma(f)$ (defined in (11.4)). We use $s^*(f)$ to represent this confidence.

$$s^*(f) = 1 - e^{-\sigma^*(f)}. \tag{11.7}$$

11.2.2.3 Handling Additional Subtlety

One problem with the above model is we have been *assuming different web sites are independent of each other*. This assumption is often incorrect because errors can be propagated between web sites. According to the definitions above, if a fact f is provided by five web sites with trustworthiness of 0.6 (which is quite low), f will have confidence of 0.99. But actually some of the web sites may copy contents from others. In order to compensate for the problem of overly high confidence, we add a *dampening factor* γ into (11.7), and redefine fact confidence as $s^*(f) = 1 - e^{-\gamma \cdot \sigma^*(f)}$, where $0 < \gamma < 1$.

The second problem with our model is that *the confidence of a fact f can easily be negative* if f is conflicting with some facts provided by trustworthy web sites, which makes $\sigma^*(f) < 0$ and $s^*(f) < 0$. This is unreasonable because even with negative evidences, there is still a chance that f is correct, so its confidence should still be above zero. Therefore, we adopt the widely used logistic function [6], which is a variant of (11.7), as the final definition for fact confidence:

$$s(f) = \frac{1}{1 + e^{-\gamma \cdot \sigma^*(f)}}. \tag{11.8}$$

When $\gamma \cdot \sigma^*(f)$ is significantly greater than zero, $s(f)$ is very close to $s^*(f)$ because $\frac{1}{1+e^{-\gamma \cdot \sigma^*(f)}} \approx 1 - e^{-\gamma \cdot \sigma^*(f)}$. When $\gamma \cdot \sigma^*(f)$ is significantly less than zero, $s(f)$ is close to zero but remains positive, which is consistent with the real situation. Equation (11.8) is also very similar to sigmoid function [7], which has been successfully used in various models in many fields.

11.2.2.4 Iterative Computation

As described above, we can infer the web site trustworthiness if we know the fact confidence and vice versa. As in Authority-hub analysis [2] and PageRank [3],

TRUTHFINDER adopts an iterative method to compute the trustworthiness of web sites and confidence of facts. Initially, it has very little information about the web sites and the facts. At each iteration TRUTHFINDER tries to improve its knowledge about their trustworthiness and confidence, and it stops when the computation reaches a stable state.

We choose the initial state in which all web sites have a uniform trustworthiness t_0. (t_0 is set to the estimated average trustworthiness, such as 0.9.) In each iteration, TRUTHFINDER first uses the web site trustworthiness to compute the fact confidence, and then recomputes the web site trustworthiness from the fact confidence. It stops iterating when it reaches a stable state. The stableness is measured by the change of the trustworthiness of all web sites, which is represented by a vector $\overrightarrow{t}$. If $\overrightarrow{t}$ only changes a little after an iteration (measured by cosine similarity between the old and the new $\overrightarrow{t}$), then TRUTHFINDER will stop.

11.2.3 A Case Study on Book Authors

In this section we present experiments on a real data set, which shows the effectiveness of TRUTHFINDER. We compare it with a baseline approach called VOTING, which chooses the fact that is provided by most web sites. We also compare TRUTHFINDER with Google by comparing the top web sites found by each of them.

This data set contains the authors of many books provided by many online bookstores. It contains 1265 computer science books published by Addison Wesley, McGraw Hill, Morgan Kaufmann, or Prentice Hall. For each book, we use its ISBN to search on *www.abebooks.com*, which returns the book information on different online bookstores that sell this book. The data set contains 894 bookstores and 34031 listings (i.e., bookstore selling a book). On average each book has 5.4 different sets of authors.

TRUTHFINDER performs iterative computation to find out the set of authors for each book. In order to test its accuracy, we randomly select 100 books and manually find out their authors. We find the image of each book and use the authors on the book cover as the standard fact.

We compare the set of authors found by TRUTHFINDER with the standard fact to compute the accuracy. For a certain book, suppose the standard fact contains x authors, TRUTHFINDER indicates there are y authors, among which z authors belong to the standard fact. The accuracy of TRUTHFINDER is defined as $\frac{z}{\max(x,y)}$.[1]

Sometimes TRUTHFINDER provides partially correct facts. For example, the standard set of authors for a book is "Graeme C. Simsion and Graham Witt," and the authors found by TRUTHFINDER may be "Graeme Simsion and G. Witt." We consider "Graeme Simsion" and "G. Witt" as partial matches for "Graeme C. Simsion" and "Graham Witt," and give them partial scores. We assign different weights to

[1] For simplicity we do not consider the order of authors in this study, although TRUTHFINDER can report the authors in correct order in most cases.

different parts of persons' names. Each author name has total weight 1, and the ratio between weights of last name, first name, and middle name is 3:2:1. For example, "Graeme Simsion" will get a partial score of 5/6 because it omits the middle name of "Graeme C. Simsion." If the standard name has a full first or middle name, and TRUTHFINDER provides the correct initial, we give TRUTHFINDER half score. For example, "G. Witt" will get a score of 4/5 with respect to "Graham Witt," because the first name has weight 2/5, and the first initial "G." gets half of the score.

The implication between two sets of authors f_1 and f_2 is defined in a very similar way as the accuracy of f_2 with respect to f_1. One important observation is that many bookstores provide incomplete facts, such as only the first author. For example, if a web site w_1 says a book is written by "Jennifer Widom," and another web site w_2 says it is written by "Jennifer Widom and Stefano Ceri," then w_1 actually supports w_2 because w_1 is probably providing partial fact. Therefore, if fact f_2 contains authors that are not in fact f_1, then f_2 is actually supported by f_1. The implication from f_1 to f_2 is defined as follows. If f_1 has x authors and f_2 has y authors, and there are z shared ones, then $imp(f_1 \rightarrow f_2) = z/x - base\_sim$, where $base\_sim$ is the threshold for positive implication and is set to 0.5.

Figure 11.4 shows the accuracies of TRUTHFINDER and VOTING. One can see that TRUTHFINDER is significantly more accurate than VOTING even at the first iteration, where all bookstores have the same trustworthiness. This is because TRUTHFINDER considers the implications between different facts about the same object, while VOTING does not. As TRUTHFINDER repeatedly computes the trustworthiness of bookstores and the confidence of facts, its accuracy increases to about 95% after the third iteration and remains stable. TRUTHFINDER takes 8.73 s to pre-compute the implications between related facts, and 4.43 s to finish the four iterations. VOTING takes 1.22 s.

Figure 11.5 shows the relative change of the trustworthiness vector after each iteration, which is defined as one minus the cosine similarity of the old and new vectors. We can see that TRUTHFINDER converges in a steady speed.

In Table 11.3 we manually compare the results of VOTING, TRUTHFINDER, and the results of the authors provided by Barnes & Noble on its web site. We list the

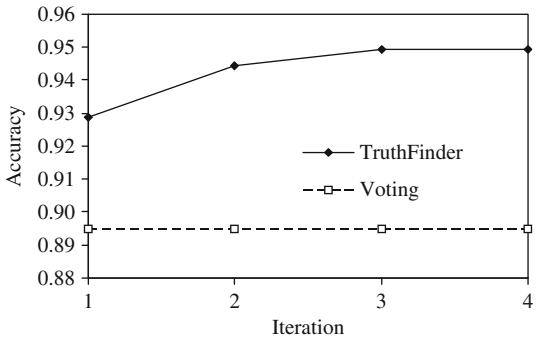


Fig. 11.4 Accuracies of TRUTHFINDER and VOTING

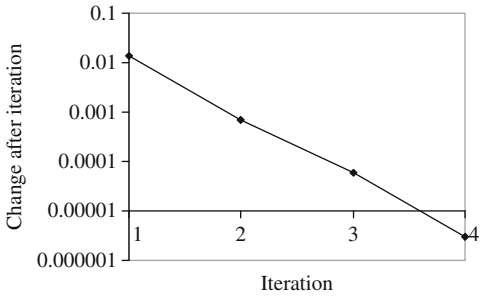


Fig. 11.5 Relative changes of TRUTHFINDER

Table 11.3 Compare the results of VOTING, TRUTHFINDER, and Barnes & Noble

Type of error	VOTING	TRUTHFINDER	Barnes & Noble
Correct	71	85	64
Miss author(s)	12	2	4
Incomplete names	18	5	6
Wrong first/middle names	1	1	3
Has redundant names	0	2	23
Add incorrect names	1	5	5
No information	0	0	2

number of books in which each approach makes each type of error. Please note that one approach may make multiple errors for one book.

VOTING tends to miss authors because many bookstores only provide subsets of all authors. On the other hand, TRUTHFINDER tends to consider facts with more authors as correct facts because of our definition of implication for book authors and thus makes more mistakes of adding in incorrect names. One may think that the largest bookstores will provide accurate information, which is surprisingly untrue. Table 11.3 shows Barnes & Noble has more errors than VOTING and TRUTHFINDER on these 100 randomly selected books.

Finally, we perform an interesting experiment on finding trustworthy web sites. It is well known that Google (or other search engines) is good at finding authoritative web sites. But do these web sites provide accurate information? To answer this question, we compare the online bookstores that are given highest ranks by Google with the bookstores with highest trustworthiness found by TRUTHFINDER. We query Google with "bookstore",[2] and find all bookstores that exist in our data set from the top 300 Google results. The accuracy of each bookstore is tested on the 100 randomly selected books in the same way as we test the accuracy of TRUTHFINDER. We only consider bookstores that provide at least 10 of the 100 books.

Table 11.4 shows the accuracy and the number of books provided (among the 100 books) by different bookstores. TRUTHFINDER can find bookstores that provide much more accurate information than the top bookstores found by Google.

[2] This query was submitted on February 7, 2007.

Table 11.4 Accuracies of top bookstores by TRUTHFINDER and by Google

TRUTHFINDER			
Bookstore	Trustworthiness	No. of Books	Accuracy
TheSaintBookstore	0.971	28	0.959
MildredsBooks	0.969	10	1.0
alphacraze.com	0.968	13	0.947
Marando.de Versandbuchhandlung	0.967	18	0.947
blackwell online	0.962	38	0.879
Annex Books	0.956	15	0.913
Stratford Books	0.951	50	0.857
movies with a smile	0.950	12	0.911
Aha-Buch	0.949	31	0.901
Players quest	0.947	19	0.936
Average accuracy			0.925
Google			
Bookstore	Google rank	No. of Books	Accuracy
Barnes & Noble	1	97	0.865
Powell's books	3	42	0.654
ecampus.com	11	18	0.847
Average accuracy			0.789

TRUTHFINDER also finds some large trustworthy bookstores, such as A1 Books (not among the top 10 shown in Table 11.4) which provides 86 of 100 books with accuracy of 0.878. Please note that TRUTHFINDER uses no training data, and the testing data are manually created by reading the authors' names from book covers. Therefore, we believe the results that suggest that there may be better alternatives than Google for finding accurate information on the web.

11.3 Distinguishing Objects with Identical Names

People retrieve information from different databases on the web, such as DBLP, Yahoo shopping, and AllMusic. One problem that has always been disturbing is that different objects may share identical names. For example, there are 197 papers in DBLP written by at least 14 different "Wei Wang"s. Another example is that there are 72 songs and 3 albums named "Forgotten" in allmusic.com. Users are often unable to distinguish them, because the same object may appear in very different contexts, and there is often limited and noisy information associated with each appearance.

In this section we study the problem of *Object Distinction*, i.e., *Distinguishing Objects with Identical Names*. Given a database and a set of references in it referring to multiple objects with identical names, our goal is to split the references into clusters, so that each cluster corresponds to one real object. We assume that the data are stored in a relational database, and the objects to be distinguished reside in a table.

At the beginning of this chapter we show an example in Fig. 11.1, which contains some papers by four different "Wei Wang"s and the linkages among them.

This problem of object distinction is the opposite of a popular problem called *reference reconciliation* (or *record linkage*, *duplicate detection*) [8], which aims at merging records with different content referring to the same object, such as two citations referring to the same paper. There have been many record linkage approaches [9–13]. They usually use some efficient techniques [14] to find candidates of duplicate records (e.g., pairs of objects with similar names), and then check duplication for each pair of candidates. Different approaches are used to reconcile each candidate pair, such as probabilistic models of attribute values [8, 12] and textual similarities [10, 11].

Compared with record linkage, objection distinction is a very different problem. First, because the references have identical names, textual similarity is useless. Second, each reference is usually associated with limited information, and thus it is difficult to make good judgment based on it. Third and most importantly, because different references to the same object appear in different contexts, they seldom share common or similar attribute values. Most record linkage approaches [8, 10–12] are based on the assumption that duplicate records should have equal or similar values, and thus cannot be used on this problem.

Although the references are associated with limited and possibly inconsistent information, the linkages among references and other objects still provide crucial information for grouping references. For example, in a publication database, different references to authors are connected in numerous ways through authors, conferences, and citations. References to the same author are often linked in certain ways, such as through their coauthors, coauthors of coauthors, and citations. These linkages provide important information, and a comprehensive analysis on them may likely disclose the identities of objects.

We develop a methodology called DISTINCT that can distinguish object identities by fusing different types of linkages with differentiating weights, and using a combination of distinct similarity measures to assess the value of each linkage. Because the linkage information is usually sparse and intertwined, DISTINCT combines two approaches for measuring similarities between records in a relational database. The first is *set resemblance between the neighbor tuples* of two records [9] (the *neighbor tuples* of a record are the tuples linked with it); and the second is *random walk probability* between two records in the graph of relational data [13]. These two approaches are complementary: one uses the neighborhood information and the other uses the connection strength of linkages.

Moreover, there are many types of linkages among references, each following a join path in the database schema. Different types of linkages have very different semantic meanings and different levels of importance. DISTINCT uses support vector machines (SVM) [15] to learn a model for weighing different types of linkages.

When grouping references, the references to the same object can be merged and considered as a whole. Therefore, DISTINCT uses agglomerative hierarchical clustering [16], which repeatedly merges the most similar pairs of clusters. It combines *Average-Link* (average similarity between all objects in two clusters) and *collective*

similarity (considering each cluster as a single object) to measure the similarity between two clusters, which is less vulnerable to noise.

The following three subsections are organized as follows. Section 11.3.1 describes the features we use to measure similarity between references. Section 11.3.2 introduces how we use machine learning approaches to combine different features and create a final measure of similarity, and Section 11.3.3 explains our approach for clustering references.

11.3.1 Similarity Between References

We say a set of references are *resembling* if they have identical textual contents (e.g., references to authors with identical names). Two references are *equivalent* if they refer to the same object, and *distinct* if they do not. Our goal is to group a set of resembling references into clusters so that there is a 1-to-1 mapping between the clusters and the real objects. In this section we describe our similarity measure for references. Because each reference is usually associated with very limited information, we utilize the relationships between each reference and other tuples in the database, with the following two types of information: (1) the *neighbor tuples* of each reference and (2) the *linkages* between different references. Based on our observation, for two references, the more overlapping on their neighborhood, or the stronger linkages between them, the more likely they are equivalent.

11.3.1.1 Neighbor Tuples

The *neighbor tuples* of a reference are the tuples joinable with it. A reference has a set of neighbor tuples along each join path starting at the relation containing references. The semantic meaning of neighbor tuples is determined by the join path. For example, in the DBLP database whose schema is shown in Fig. 11.6, we study the references to authors in $Publish$ relation. For a particular reference in $Publish$ relation, its neighbor tuples along join path "$Publish \bowtie Publications \bowtie Publish \bowtie Authors$" represent the authors of the paper for this reference. Because different join paths have very different semantic meanings, we treat the neighbor tuples along each join path separately, and will combine them later by supervised learning.

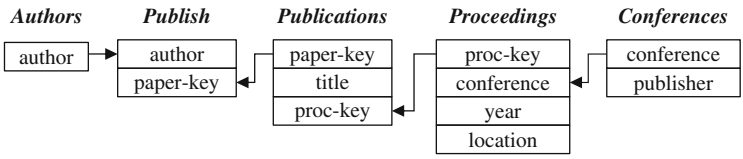


Fig. 11.6 The schema of DBLP database

Definition 3 (Neighbor tuples) Suppose the references to be resolved are stored in relation R_r. For a reference r that appears in tuple t_r, and a join path P that starts at

R_r and ends at relation R_t, the neighbor tuples of r along join path P, $NB_P(r)$, are the tuples in R_t joinable with t_r along P.

Besides neighbor tuples of each reference, the attribute values of neighbor tuples are also very useful for reconciling references. For example, two neighbor tuples in *Conferences* relation sharing the same value on *publisher* attribute indicate some relationship between these two tuples. In DISTINCT, *we consider each value of each attribute (except keys and foreign-keys) as an individual tuple*. For example, each distinct value of *publisher* attribute (ACM, Springer, etc.) is considered as a tuple, and the *publisher* attribute in *Proceedings* relation is considered as a foreign-key referring to those tuples. In this way we can use a single model to utilize both neighbor tuples and their attribute values.

11.3.1.2 Connection Strength

For a reference r and a join path P, the strengths of relationships between r and different tuples in $NB_P(r)$ could be very different (e.g., the different relationships between an author and different coauthors). We use probability propagation [17] to model the connection strength between a reference r and its neighbor tuples $NB_P(r)$. Initially the tuple containing r has probability 1. At each step, for each tuple t with non-zero probability, we uniformly propagate t's probability to all tuples joinable with t along the join path P. For each tuple $t \in NB_P(r)$, we compute $Prob_P(r \to t)$, the probability of reaching t from r via join path P, which is used as the *connection strength* between r and t. We also compute $Prob_P(t \to r)$, which is the probability of reaching r from t via the reverse join path of P.

The computation of both types of probabilities can be done in a depth-first traversal of all qualified join paths. Figure 11.7 shows the procedure of propagating probabilities from a tuple in R_r to tuples in R_1 and R_2. The two numbers in each box are the probability of reaching this tuple and the probability of reaching the origin from this tuple.

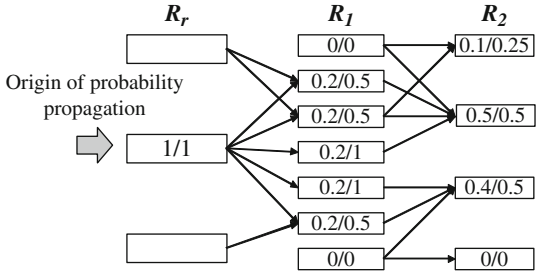


Fig. 11.7 Propagating probabilities between tuples

11.3.1.3 Set Resemblance of Neighbor Tuples

Our first measure for similarity between references is the set resemblance between their neighbor tuples, which represents the similarity between the contexts of two references in a relational database. The set resemblance between neighbor tuples is defined by Jaccard coefficient [18]. Because a reference has different connection strengths to different neighbor tuples, we use such strengths as weights in Jaccard coefficient.

Definition 4 (Set Resemblance) The set resemblance similarity between two references r_1 and r_2 with respect to join path P is defined as

$$Resem_P(r_1, r_2) = \frac{\sum_{t \in NB_P(r_1) \cap NB_P(r_2)} \min(Prob_P(r_1 \to t), Prob_P(r_2 \to t))}{\sum_{t \in NB_P(r_1) \cup NB_P(r_2)} \max(Prob_P(r_1 \to t), Prob_P(r_2 \to t))}.$$

11.3.1.4 Random Walk Probability

Another important factor for similarity between references is the linkages between them. We use the random walk probability model used in multi-relational record linkage [13]. The total strength of the linkages between two references is defined as the probability of walking from one reference to the other within a certain number of steps. A distinguishing feature of our approach is that we compute the random walk probability along each join path separately, so as to acknowledge the different semantics of different join paths.

It is usually expensive to compute random walk probabilities along long join paths. Since we have computed the probabilities of walking from references to their neighbor tuples, and those from neighbor tuples to references, we can easily compute the probability of walking between two references by combining such probabilities.

In general, random walk probability indicates the linkage strength between references. It is complementary to set resemblance, which indicates the context similarity of references. DISTINCT combines both measures to perform comprehensive analysis on similarities between references.

11.3.2 Supervised Learning with Automatically Constructed Training Set

In previous record linkages approaches that utilize relation information [9, 13], all join paths are treated equally. However, linkages along different join paths have very different semantic meanings, and thus should have different weights. For example, in DBLP database two references to authors being linked by the same coauthor is a strong indication of possible equivalence, whereas two references being linked by the same conference is much weaker.

DISTINCT uses supervised learning to determine the pertinence of each join path and assign a weight to it. In order to do this, a training set is needed that contains

equivalent references as positive examples and distinct references as negative ones. Instead of manually creating a training set which requires much labor and expert knowledge, DISTINCT constructs the training set automatically, based on the observation that the majority of entities have distinct names in most applications. Take the problem of distinguishing persons as an example. A person's name consists of the first and last names. If a name contains a rather rare first name and a rather rare last name, this name is very likely to be unique. We can find many such names in a database and use them to construct training sets. A pair of references to an object with a unique name can be used as a positive example, and a pair of references to two different objects can be used as a negative example.

Given the training examples, we use support vector machines (SVM) [15] to learn a model based on similarities via different join paths. We introduce the learning procedure for set resemblance similarities, and the same procedure is also applied on random walk probabilities. Each training example (which is a pair of references) is converted into a vector, and each dimension of the vector represents set resemblance similarity between the two references along a certain join path. Then SVM with linear kernel is applied to these training sets, and the learned model is a linear combination of similarities via different join paths. Usually some important join paths have high positive weights, whereas others have weights close to zero and can be ignored in further computation. Let $Resem(r_1, r_2)$ be the overall set resemblance similarity between r_1 and r_2:

$$Resem(r_1, r_2) = \sum_{P \in \mathcal{P}} w(P) \cdot Resem_P(r_1, r_2), \tag{11.9}$$

where $w(P)$ is the weight of join path P.

11.3.3 Clustering References

Given a set of references to the same name, DISTINCT tries to group them into clusters, so that each cluster corresponds to a real entity. The procedure of clustering references will be discussed in this section.

11.3.3.1 Clustering Strategy

The problem of clustering references has the following special features: (1) the references do not lie in a Euclidean space, (2) the number of clusters is completely unknown, and (3) equivalent references can be merged into a cluster, which still represents a single object. Therefore, agglomerative hierarchical clustering is most suitable for this problem, as it first uses each reference as a cluster, and then repeatedly merges the most similar pairs of clusters.

A most important issue is how to measure the similarity between two clusters of references. There are different measures including Single-Link, Complete-Link, and Average-Link [16]. Because references to the same object may form weakly linked

partitions, Complete-Link is not appropriate. On the other hand, references to different objects may be linked, which make Single-Link inappropriate. In comparison, Average-Link is a reasonable measure, as it captures the overall similarity between two clusters and is not affected by individual linkages which may be misleading.

Average-Link still suffers from the problem that references to the same object often from weakly linked partitions. For example, in DBLP an author may collaborate with different groups of people when he/she is affiliated with different institutions. When these partitions are large, the Average-Link similarity may be small even if there are many linkages between them. To address this problem, we combine Average-Link with the *collective random walk probability* between two clusters, which is the probability of walking from one cluster of references to the other cluster. In details, we adopt a composite similarity measure by combining the average set resemblance similarity with the collective random walk probability when measuring similarity between clusters. Because these two measures may have different scales, and arithmetic average will often ignore the smaller one, we use the geometric average of the two measures as the overall similarity between two clusters:

$$Sim(C_1, C_2) = \sqrt{Resem(C_1, C_2) \cdot WalkProb(C_1, C_2)}, \tag{11.10}$$

where $Resem(C_1, C_2)$ is the average set resemblance similarity between references in C_1 and those in C_2, and $WalkProb(C_1, C_2)$ is the collective random walk probability between them.

11.3.3.2 Computing Clusters

Initially each reference is used as a cluster, and the set resemblance similarity and random walk probability between each pair of clusters are computed. This is usually affordable because the number of references having identical names is seldom very large. At each step, the two most similar clusters C_1 and C_2 are merged into a new cluster C_3, and we need to compute the similarity between C_3 and each existing cluster C_i. When C_3 is very large, a brute-force method may consume similar amount of time as computing pair-wise similarity during initialization, and it is unaffordable to perform such computation at every step.

To address this problem, we design efficient methods that can incrementally compute the similarity between clusters as clusters are merged. One important observation for improving efficiency is that both the average set resemblance similarity and random walk probability between C_3 and C_i can be directly computed by aggregating the similarities between C_1, C_2, and C_i.

11.3.4 A Case Study on Authorship on DBLP

We report our empirical study on testing the effectiveness of the proposed approach. DISTINCT is tested on DBLP database, whose schema is shown in Fig. 11.6. First,

authors with no more than two papers are removed, and there are 127,124 authors left. There are about 616K papers and 1.29M references to authors in *Publish* relation (authorship). In DBLP we focus on distinguishing references to authors with identical names.

When computing set resemblance of neighbor tuples (Section 11.3.1.3), we consider neighbor tuples reached by join paths of at most four joins. When computing random walk probability (Section 11.3.1.4), we consider join paths of at most eight joins.

We first build a training set using the method in Section 11.3.2, which contains 1000 positive and 1000 negative examples. Then SVM with linear kernel is applied. The whole process takes 62.1 s. We measure the performance of DISTINCT by precision, recall, and f-measure. Suppose the standard set of clusters is C^* and the set of clusters by DISTINCT is C. Let TP (true positive) be the number of pairs of references that are in the same cluster in both C^* and C. Let FP (false positive) be the number of pairs of references in the same cluster in C but not in C^*, and FN (false negative) be the number of pairs of references in the same cluster in C^* but not in C:

$$precision = \frac{TP}{TP + FP}, \qquad recall = \frac{TP}{TP + FN}.$$

f-measure is the harmonic mean of precision and recall.

We test DISTINCT on real names in DBLP that correspond to multiple authors. Ten such names are shown in Table 11.5, together with the number of authors and number of references. For each name, we manually divide the references into groups according to the authors' identities, which are determined by the authors' home pages or affiliations shown on the papers.[3]

Table 11.5 Names corresponding to multiple authors

Name	No. of Authors	No. of References	Name	No. of Authors	No. of References
Hui Fang	3	9	Bing Liu	6	89
Ajay Gupta	4	16	Jim Smith	3	19
Joseph Hellerstein	2	151	Lei Wang	13	55
Rakesh Kumar	2	36	Wei Wang	14	141
Michael Wagner	5	29	Bin Yu	5	44

We use DISTINCT to distinguish references to each name, with min-sim set to 0.0005. Table 11.6 shows the performance of DISTINCT for each name. In general, DISTINCT successfully group references with high accuracy. There is no false positive in seven out of ten cases, and the average recall is 83.6%. In some cases

[3] References whose author identities cannot be found (e.g., no electronic version of paper) are removed. We also remove authors with only one reference that is not related to other references by coauthors or conferences, because such references will not affect accuracy.

Table 11.6 Accuracy for distinguishing references

Name	Precision	Recall	F-measure
Hui Fang	1.0	1.0	1.0
Ajay Gupta	1.0	1.0	1.0
Joseph Hellerstein	1.0	0.810	0.895
Rakesh Kumar	1.0	1.0	1.0
Michael Wagner	1.0	0.395	0.566
Bing Liu	1.0	0.825	0.904
Jim Smith	0.888	0.926	0.906
Lei Wang	0.920	0.932	0.926
Wei Wang	0.855	0.814	0.834
Bin Yu	1.0	0.658	0.794
Average	0.966	0.836	0.883

references to one author are divided into multiple groups. For example, 18 references to "Michael Wagner" in Australia are divided into two groups, which lead to low recall.

We compare six versions: (1) DISTINCT, (2) DISTINCT without supervised learning, and (3–6) DISTINCT using each of the two similarity measures: set-resemblance [9] and random walk probabilities [13] (with and without supervised learning). Note that supervised learning is not used in [9] and [13]. For each approach except DISTINCT, we choose the min-sim that maximizes average accuracy. Figure 11.8 shows the average f-measure of each approach. DISTINCT leads by about 15% compared with the approaches in [9] and [13]. The f-measure is improved by more than 10% with supervised learning, and 3% with combined similarity measure.

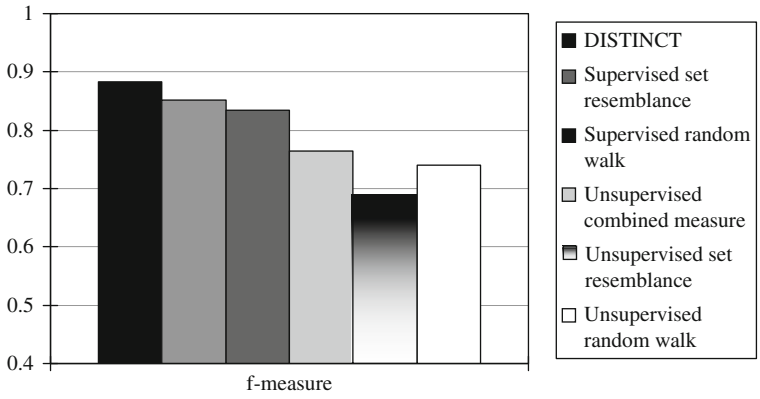


Fig. 11.8 Accuracy and f-measure on real cases

We visualize the results about "Wei Wang" in Fig. 11.9. References corresponding to each author are shown in a gray box, together with his/her current affiliation and number of references. The arrows and small blocks indicate the mistakes made by DISTINCT. It can be seen that in general DISTINCT does a very good job in

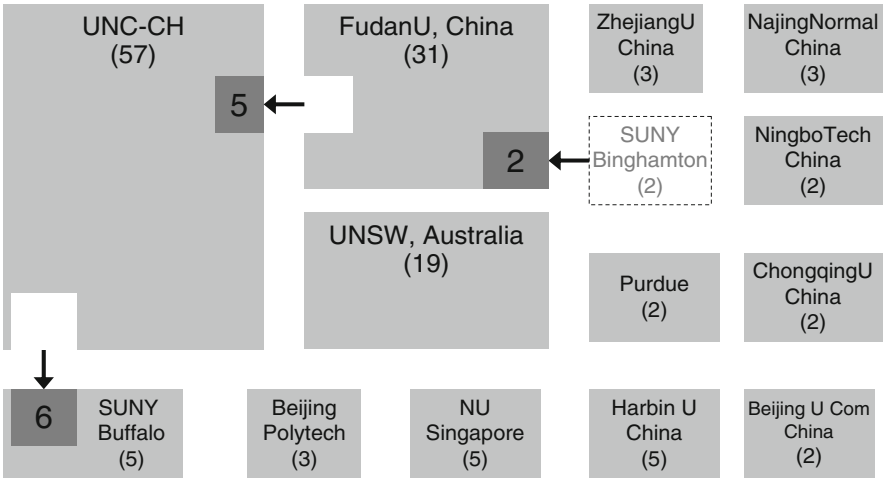


Fig. 11.9 Groups of references of "Wei Wang"

distinguishing references, although it makes some mistakes because of the linkages between references to different authors.

11.4 Conclusions

In this chapter we study two problems about correctness of information on the web. We first introduce and formulate the Veracity problem, which aims at resolving conflicting facts from multiple web sites, and finding the true facts among them. We propose TRUTHFINDER, an approach that utilizes the interdependency between web site trustworthiness and fact confidence to find trustable web sites and true facts.

Then we study the problem of distinguishing references to people or objects with identical names. We develop a general methodology called DISTINCT for supervised composition of heterogeneous link analysis that can fuse different types of linkages and use a combination of distinct similarity measures to assess the value of each linkage.

Our experiments show that TRUTHFINDER achieves high accuracy at finding true facts and at the same time identifies web sites that provide more accurate information, and DISTINCT can accurately distinguish different objects with identical names in real databases. In general, this chapter shows that linkages can help us analyze the veracity and identity of people, objects, and their information.

Both TRUTHFINDER and DISTINCT are limited to time-invariant data. An interesting research direction is how they can be adapted to handle the changes of data over time. It is also interesting to study how to handle entities with multiple forms of names, which exist in many domains.

References

1. Princeton Survey Research Associates International. Leap of faith: Using the Internet despite the dangers. Results of a National Survey of Internet Users for Consumer Reports WebWatch, Oct 2005.
2. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *SODA*, pages 668–677, San Francisco, CA, 1998.
3. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
4. A. Borodin, G. Roberts, J. Rosenthal, and P. Tsaparas. Link analysis ranking: Algorithms, theory, and experiments. *ACM Transactions on Internet Technology*, 5(1):231–297, 2005.
5. X. Yin, J. Han, and P. S. Yu. Truth discovery with multiple conflicting information providers on the Web. *IEEE Transaction on Knowledge and Data Engineering*, 20(6):796–808, 2008.
6. Logistical Equation from Wolfram MathWorld. http://mathworld.wolfram.com/LogisticEquation.html, Accessed on 2009/08/01.
7. Sigmoid Function from Wolfram MathWorld. http://mathworld.wolfram.com/SigmoidFunction.html, Accessed on 2009/08/01.
8. W. Winkler. The State of Record Linkage and Current Research Problems. Stat. Research Div., U.S. Bureau of Census, 1999.
9. I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. In *MRDM workshop*, Chicago, IL, 2005.
10. M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, pages 39–48, Washington, DC, 2003.
11. S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, pages 313–324, San Diego, CA, 2003.
12. I. Felligi, and A. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64(328):1183–1210, 1969.
13. D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SDM*, pages 262–273, Newport Beach, CA, 2005.
14. L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, Trondheim, Norway, 2001.
15. C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–168, 1998.
16. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31:264–323, 1999.
17. Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000.
18. P. N. Tan, M. Steinbach, and V. Kumar. Introduction to Data Mining. Addison-Wesley, Boston, MA, 2005.

Part IV
Social Network Analysis

Chapter 12
Dynamic Community Identification

Tanya Berger-Wolf, Chayant Tantipathananandh, and David Kempe

Abstract Humans and other social animals interact in diverse ways. A central problem in the study of societies is identifying core communities: sets of entities among whom interactions are frequent and consistent. Membership in social groups often changes, thus making it difficult to characterize a society's community structure. In this chapter we formalize the computational problem of dynamic community identification and review computational methods that address the changing nature of community membership. We discuss in detail the dynamic community identification method by the authors which in many ways subsumes other approaches.

12.1 Introduction

Social creatures interact in diverse ways: forming groups, sending emails, sharing ideas, and mating. Some of the interactions are accidental while others are a consequence of the underlying explicit or implicit social structures [10, 15, 34]. In order to understand social interactions, it is therefore crucial to identify these social structures or "communities," which are loosely defined as collections of individuals who interact unusually frequently [20, 21, 23, 25, 35, 58]. Community structure often reveals interesting properties shared by the members, such as common hobbies, occupations, social functions, or rank [6, 10, 15]. The importance of community structure extends beyond social networks in the strictest sense: for instance, in hyperlinked documents such as the World Wide Web, salient properties of "individuals" (in this case, pages) include related topics or common viewpoints. As a result, there has been a large amount of research on identifying communities in the web graph or similar settings (see Chapter 7).

Perhaps one of the most interesting aspects of community structure is its evolution over time: how do membership and interactions change, and how does this change correlate with properties of the individuals or events within the time frame [2, 26, 30, 38, 45–47, 56]? Is the turnover of groups fast or slow? Do they

T. Berger-Wolf (✉)
University of Illinois at Chicago, Chicago, IL 60607, USA
e-mail: tanyabw@uic.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_12,

assemble and disband spontaneously or over a long period of time? With improvements in technology, collecting longitudinal data is becoming increasingly feasible, in turn allowing us to begin answering these questions systematically. However, in addition to data, this task also requires novel algorithms which can not only identify community structure but also track its changes over time. In this chapter, we investigate models and algorithms for dynamic community identification. A particular focus will be given to recent work by the authors, which derives a measure of parsimony of a community interpretation from first principles about the interactions of individuals.

12.2 Static and Dynamic Networks

Community identification in static networks (that do not change over time) is a very well studied topic. Some of the approaches date back to the 1930s [10]. With the ascent of computation as a scientific tool, the last 10–15 years have seen an explosion in different algorithms for community identification. These algorithms are based on different precise formalization of the vague concept of "unusually frequent interactions"; they also differ in their efficiency and many other parameters, making them useful in different settings. For several recent comparative surveys, see, for example, [9, 17, 18, 21, 39]. Chapter 7 specifically addresses the problem of community identification and analysis in the *web graph*.

For static networks, the notion of a "community" is at least intuitively clear. When analyzing dynamically changing networks, such as could result from a longitudinal study of a population, even the very nature of what we mean by a community becomes less clear. On the one hand, a community should be *persistent*, i.e., the interactions among its members should be observed over a period of time. On the other hand, the interactions within a community also need to be more frequent than with the rest of the network. Clearly, these two objectives may be in conflict with each other.

One may be tempted to leverage the vast body of work on communities in static graphs by doing one of the following: (1) focus on one particular point in time or (2) aggregate the sequence of observations (or a subsequence) into one graph, as a union of edges, possibly with weights. The first approach obviously discards any information about the evolution of relationships over time and further raises the issue of how to choose the right point in time. The second approach, frequently used by practitioners (e.g., [3, 4, 21, 33, 60]), has problems of its own. As an example, consider the static network in Fig. 12.1, which was derived by aggregating a sequence of dynamic networks. Figure 12.2 shows four possible sequences of networks which may have yielded the network in Fig. 12.1 in this way. Any static community detection algorithm would thus output the same communities for all four input sequences, even though the sequences are manifestly different. Thus, it is crucial to focus on the *entire sequence*, as a sequence, and discover the dynamic change affecting the communities over time.

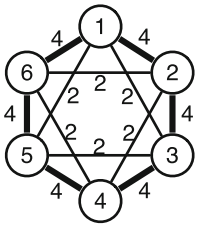
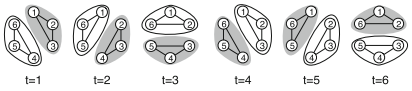
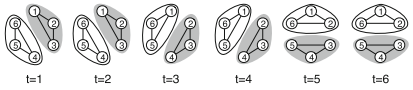


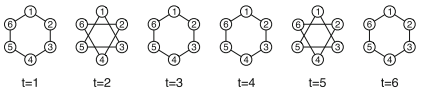
Fig. 12.1 An example static (aggregated) network representation. The *circles* are nodes corresponding to individual entities, and *lines* are links (or edges) connecting those entities that have interacted at any time. The thickness of a line and numbers on each line indicate the frequency of interaction between a joined pair



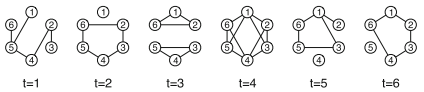
(a) Dynamic affiliation network with fast turnover.



(b) Dynamic affiliation network with slow turnover.



(c) Periodic non-affiliation network.



(d) Random dynamic network.

Fig. 12.2 Four out of many possible dynamic networks which when aggregated result in the same static network of Fig. 12.1. Each panel (**a**–**d**) shows alternative possible interaction scenarios among the same six nodes (labeled 1–6) over six time steps ($t = 1, \ldots, 6$). In (**a**) and (**b**), the dynamic networks are affiliation networks. The *white* and *shaded* envelopes around the nodes indicate the groups observed at each time step. The difference between scenario (**a**) and scenario (**b**) can be viewed as the order in which groups are observed or the rate of turnover. In (**c**), nodes interact pairwise and no groups can be seen. Thus, (**c**) is not an affiliation network. It is, however, regular and periodic. Network (**d**) is a random assignment of edges to time steps. It is not an affiliation network since interactions are not transitive

Intuitively, when we talk about "dynamic networks," we will consider a time series of graphs (a more formal definition will be given in Section 12.4). A graph in this series captures all interactions observed between the individuals over a small time window, such as a day or an hour.[1]

For much of this overview, we will be focusing on a special class of social networks termed *affiliation networks* by social scientists [6]. Affiliation networks are collapsed bipartite graphs of individuals and their affiliations. Thus, in an affiliation network, the nodes represent individuals and the edges represent a shared affiliation, such as membership in an organization, or a physical presence at a meeting. We assume that at any given time, each individual can only be affiliated with one group

[1] The appropriate definition of a time step is nontrivial and beyond the scope of this chapter. Yet we note that many interaction systems lend themselves to natural time quantizations, due to the behavioral patterns of individuals, or observational methods.

(although affiliations can change over time). Affiliation is a transitive relationship; therefore, all individuals sharing an affiliation form a clique.[2] An affiliation network is, thus, a collection of disjoint cliques (which may be different from the underlying community structure). Recently, Lattanzi and Sivakumar [31] have proposed the first computational model for a *growing* affiliation network.[3] In the spirit of the previous discussion, a *dynamic affiliation network* is a time series of affiliation networks. Figure 12.2a–b shows two dynamic affiliation networks. Figure 12.2c–d, however, are not affiliation networks, because interactions are not transitive.

12.3 Overview of Dynamic Community Identification Methods

In this overview, we focus on methods which aim to infer latent community structure purely from the observed interactions, without recourse to additional information about the individuals (such as age, gender, location, interests). Thus, the input to these methods is a dynamic social network or a dynamic affiliation network. Different methods take different approaches toward quantifying the relationship between the individual interactions and the underlying community structure.

Standard definitions of communities in social networks rely on various measures of cohesiveness [36, 58, Section 12.7.6]. Communities are loosely defined as sets of individuals more closely affiliated with each other than with members of other communities. Quantifying the notion of "close affiliation" over time is, however, nontrivial. Consider the examples of Fig. 12.2: the three distinct dynamic scenarios all lead to the same static network, the one shown in Fig. 12.1. In the static case, any community identification method would identify only a single community. By contrast, in the dynamic cases of (a) and (b), we can see that there exist individuals in two groups that change over time. Two groups of individuals, starting with $\{1, 2, 3\}$ and $\{4, 5, 6\}$, exchange one individual at a time. In example (a), this exchange occurs in each time step, while in example (b), this exchange happens only every other time step. These examples are abstract representations of two distinct groups gradually exchanging their membership, and there are many real phenomena that can be viewed in this way. The key to the existence of two distinct communities here is the *gradual* nature of exchange which ensures the continuity and persistence of each community over time.

Perhaps the earliest method to directly address the changing nature of groups over time is by Aggarwal and Yu [1]. They proposed a method to track groups that change (grow or shrink) faster than their surroundings. The method does not infer communities but rather tracks the gradual changes of a given community in time.

The notion of continuity and persistence is most directly expressed in the metagroups approach of Berger-Wolf and Saia [5]. The paper proposes a framework for tracking the evolution of groups over time in a dynamic affiliation network. The

[2] A *clique* or *complete graph* has an edge between every pair of nodes.

[3] For models for general dynamic (longitudinal) social networks, see [24, 45–47].

authors propose a representation by a directed multipartite graph with nodes representing groups of affiliation in each time step and edges connecting groups that have a significant overlap in their membership. Significance of overlap is defined in terms of a threshold chosen by the user. The edges are oriented forward in time. A dynamic community—also called a "metagroup" by the authors—is then a long path in this graph. Again, the precise required length is specified by the user. With its reliance on several user-defined parameters, the approach is again largely focused on tracking dynamic communities rather than inferring them.

A similar approach was proposed by Spiliopoulou et al. [49]. It also assumes as input a time series of partitions of individuals into groups and matches groups across time steps based on membership overlap and an added notion of time decay. However, in this model, groups are allowed to be matched to at most one (most similar) group in a later time step. One can then again construct a graph with nodes representing groups in adjacent time steps and edges capturing similarities between groups. As a result, dynamic communities are, again, paths. Depending on the relative properties of the matched groups, the transitions are classified as size or compactness change. A life history of a dynamic community can then be characterized in terms of these transitions. Like the metagroups approach, the method is better suited for tracking communities than for inferring them.

Two methods apply the idea of local matching of groups across time to general dynamic networks. Palla et al. define a community as a set of overlapping cliques [37] (non-disjoint affiliations). In taking the temporal information into consideration, the approach finds cliques of a specific size in the union of snapshot graphs of two consecutive time steps. Although communities found show their persistence, and the members of the communities can change over time, growing or shrinking is not allowed: each snapshot of a community has to be a clique of a specific fixed size. Moreover, the focus on consecutive time steps is too local and essentially memoryless. Falkowski et al. [13, 14] use the algorithm of Girvan and Newman [23] to identify snapshot communities. They then construct a graph of snapshot communities, where edges represent possible traces of communities. They then apply the algorithm of Girvan and Newman once more on this graph to find dynamic communities.

A very different approach is proposed by Sun et al. [50]. Their method uses an information-theoretic criterion to detect dramatic changes in the time series of graphs. In particular, the approach uses the principle of minimum description length to compress the consecutive network snapshots. It thus identifies communities which are locally static. Although the approach in [50] detects points of dramatic changes in community structure, it does not trace gradual changes.

Two recent methods propose a generative Markov process to model dynamic communities and find community structure maximizing the likelihood under the proposed model. Using a Bayesian approach, Lin et al. [32] propose FacetNet, which models dynamic communities as a hidden Markov chain. They maximize the posterior distribution using the EM algorithm. Yang et al. [59] propose a different Markov process and use simulated annealing to find community structure which maximizes the likelihood of the interaction data. Both methods take only local

changes into consideration. Neither approach uses the models of group dynamics from social sciences [38]. A somewhat related idea is used by Tong et al. [55] who use a low-rank approximation of adjacency matrices to infer dynamic communities.

The approach of modeling dynamic social networks via a hidden Markov model is also taken in recent papers by Sarkar and Moore [42] and Fu et al. [22]. Both of these papers place assumptions on the movement of individuals in a latent space and treat the sequence of networks as a manifestation of the individuals' positions. They use different Bayesian inference algorithms to infer the trajectory of individuals in the latent space and, thus, learn about the roles of individuals. While not explicitly addressing the issue of communities and their turnovers, the individuals' positions can be considered indicative of community structure as well.

Much of the remainder of this survey is based on two papers by the authors [53, 54]. The approach taken there is to derive an objective from first principles of social behavior. Specifically, the goal is to axiomatize the persistence of social interactions and their dynamics, in a way that explicitly draws a connection between latent community structure and observed interactions. Based on this axiomatization, we formulate a combinatorial optimization problem. Dynamic communities are essentially viewed as dynamic clusters, where some notion of "social cost" is minimized within communities. The optimization problem itself is computationally intractable; we therefore analyze various methods for speeding up the computation, including a recent approximation algorithm [53].

12.4 Dynamic Community Identification Problem

We now formally state the definitions and notations that will be used throughout the chapter and define the computational dynamic community identification optimization problem.

12.4.1 Notation and Definitions

Let $X = \{x_1, \ldots, x_n\}$ be the set of n individuals observed over T discrete time steps $t = 1, \ldots, T$. Let $H_t = \{g_{j,t}\}$ be a collection of groups of the individuals observed at time t. The interpretation is that the individuals in group $g_{j,t} \subseteq X$ are observed interacting among themselves at time t (i.e., forming a clique).

An *interaction sequence* of individuals over T time steps is $\mathcal{H} = \langle H_1, H_2, \ldots, H_T \rangle$. A *dynamic affiliation network* $(X, \mathcal{H})$ is the set of individuals and their interaction sequence over T time steps. We use the terms "interaction sequence" and "dynamic affiliation network" interchangeably.

We stress that in our terminology, groups and communities are different concepts: groups are considered as observed or input data which capture only a snapshot of interactions at one point in time, while communities are latent concepts which exist over time and should explain many of the actual observed interactions, though not necessarily all of them.

12.4.2 Problem Formulation

For a society's history of interactions, we aim to identify the most *parsimonious* underlying communities. The inferred community structure should explain as many of the observed interactions as possible. Stated conversely, we seek the community structure that minimizes those interactions considered to be among individuals in separate communities. This view is motivated by the intuitive understanding of a community as a body which exists for a certain time period, during which it has consistent membership and welcomes few outsiders.

Just as static communities are essentially clusters of individuals among whom the social distance (as expressed by the amount and frequency of interaction [20, 21, 23, 25, 35, 58]) is minimized, so, too, dynamic communities can be viewed as dynamic clusters. The question, then, is how to quantify the notion of social distance in dynamic networks. Here, we propose to measure that distance in terms of a social cost incurred by individuals in various contexts of interactions. Our definition of social cost is based on two explicit assumptions about individual behavior, motivated by research in social sciences and social physiology. First, we assume that *individuals tend not to change their home community affiliation too often* [2]. Second, we assume that *individuals tend to interact with their respective home communities most of the time* [58, p. 320]. In human and animal societies, individuals incur real social costs when their behavior deviates from these assumptions. Consider the example of a typical social club. Becoming a permanent member of the club involves a large fee and initiation rites. Attending a members-only gathering, as an outsider, results in a higher entry fee. Once one is a member, being absent from a meeting represents a wasted expenditure and may even bring on a penalty. For a less formal situation, but with similar individual behavior, consider a circle of friends. To become integrated within the circle, a newcomer typically spends time and energy getting to know the current members. Friends enjoy being with each other and are loath to miss a get-together.

With these intuitive examples in mind, we define three cost parameters potentially incurred by an individual. First, we posit a cost for a *switch* of a home community. Second, there is a cost of *visiting* a community of which one is not a member. Third, in data sets for which not all individuals are observed all the time, we have a cost of *absence*. This cost accounts for an individual who is a member of a particular community, but is absent from an observed home community gathering. Now, we can define a dynamic community, analogously to a static one, as a series of sets of individuals among whom the overall social cost of interacting is lower relative to the interactions among individuals across communities.

Definition 1 Given the three costs of switching, visiting, and absence, the DYNAMIC COMMUNITY IDENTIFICATION problem (DCI) is the problem of finding the community structure, fitting the observed pattern of interactions, for which the overall social cost is minimized.

12.4.3 Dynamic Community Identification Optimization Problem

Just like there are hundreds of formulations of the classical clustering problem, there are many ways to instantiate DYNAMIC COMMUNITY IDENTIFICATION as an optimization problem. We now present one of the possible implementations which aims to minimize the sum of non-interactions within communities and the interactions across.[4] To aid the mathematical formulation of the problem, we make the explicit technical assumption that in each time step, every group is a representative of a distinct community. If two groups are present at the same time, there is a reason they are separate: they are gatherings of individuals that chose to be apart.

The DYNAMIC COMMUNITY IDENTIFICATION problem can be intuitively modeled as a graph coloring problem [27] (albeit with a different objective function from traditional graph coloring).

For a dynamic affiliation network $(X, \mathcal{H} = \langle H_t \rangle)$, we create a *cost graph* G with one *individual vertex* $v_{i,t}$ for every individual $x_i \in X$ and every time $t = 1, \ldots, T$. In addition, there is one *group vertex* $u_{g,t}$ for every group $g \in H_t$. There are three kinds of edges:

- E_{sw} (switch): For each individual x_i and time $t \leq T - 1$, there is an edge between $v_{i,t}$ and $v_{i,t+1}$.
- E_{vis} (visit): For each individual $x_i \in g$ at time t, there is an edge between $v_{i,t}$ and $u_{g,t}$.
- E_{ab} (absence): For each individual $x_i \in X$ and each group g at time t such that $x_i \notin g$, there is an edge between $v_{i,t}$ and $u_{g,t}$.

Figure 12.3 shows an example of an interaction sequence (left) and the corresponding graph representation (middle) with all the E_{sw} edges and some of the E_{vis}, E_{ab} edges. The fragment on the right shows the graph representation of time step T3 with all the edges E_{vis}, E_{ab} present. Notice that each time step t corresponds to a complete bipartite subgraph with edges connecting only the individual vertices $v_{i,t}$ and the group vertices $u_{g,t}$ at time t.

We investigate a restricted case of the DYNAMIC COMMUNITY IDENTIFICATION problem. We constrain the individuals to belong to only one community at any given time t. This is another technical constraint to aid the formulation of the problem which should be relaxed for a general solution. Since communities are different from groups, this restriction is not captured by the definition of affiliation networks, where each individual belongs to only one *group* at any time.

Definition 2 A *community interpretation* of a dynamic affiliation network $(X, \mathcal{H})$ is a vertex coloring $\chi : V \to \mathbb{N}$ of the corresponding cost graph $G = (V, E)$. The color of an individual vertex $v_{i,t}$ represents the individual x_i's community affiliation at time t. The color of a group vertex $u_{g,t}$ gives the community that g represents at

[4] An alternative, non-equivalent, formulation, for example, would be to *maximize* the interactions within communities and non-interactions across, or some combination of those four objectives.

time t. We say that the coloring is *valid* if, for every time step t, the group vertices $u_{g,t}$ for all $g \in H_t$ have different colors.

To measure the quality of a community interpretation χ, we use the three different social costs, *switch*, *visit*, and *absence*, given as input parameters c_{sw}, c_{vis}, and c_{ab}. We note that costs are incurred by individuals.

Switch: A cost of c_{sw} is incurred if an individual switches community affiliation; that is, $\chi(v_{i,t}) \neq \chi(v_{i,t+1})$. In other words, the end points of an edge in E_{sw} (connecting an individual to itself) have different colors.

Visit: A cost of c_{vis} is incurred if an individual visits another community. That is, $x_i \in g$ (and $g \in H_t$), but $\chi(v_{i,t}) \neq \chi(u_{g,t})$. In other words, the end points of an edge in E_{vis} (connecting an individual to its current group) have different colors.

Absence: A cost of c_{ab} is incurred if an individual is absent from its community. That is, $\chi(v_{i,t}) = \chi(u_{g,t})$ but $x_i \notin g$. In other words, the end points of an edge in E_{ab} (connecting an individual to groups other than its current within the time step) have the same color.

Thus, if an individual x_i and a group g are both present and have the same color at time t, but x_i is not a member of g at that time, then the individual incurs both the visiting and absence costs. The first cost penalizes the individual for being different from its current group (which is not g and, in a valid coloring, must have a different color from g) while the second cost penalizes the individual for being absent from its current community.

The DYNAMIC COMMUNITY IDENTIFICATION *optimization* problem is then to find a valid community interpretation minimizing the total cost resulting from the switches, visits, and absences of individuals.

Definition 3 Given a dynamic affiliation network $(X, \mathcal{H})$, the social costs c_{sw}, c_{vis}, c_{ab}, and the corresponding cost graph G, let χ be a valid coloring of G, and let $\chi_{ww'}$ be an indicator variable which equals 1 if vertices w and w' have the same color in χ, and 0 otherwise. The DYNAMIC COMMUNITY IDENTIFICATION *optimization* problem is then to find a valid coloring χ of G which minimizes the total cost,

$$c_{\text{sw}} \sum_{(w,w') \in E_{\text{sw}}} (1 - \chi_{ww'}) + c_{\text{vis}} \sum_{(w,w') \in E_{\text{vis}}} (1 - \chi_{ww'}) + c_{\text{ab}} \sum_{(w,w') \in E_{\text{ab}}} \chi_{ww'}.$$

Once such a coloring χ has been found, we identify each *community* C_c with the set of groups $\{g\}$ of color $\chi(u_{g,t}) = c$. In a valid coloring, a community will contain at most one group from each time step. The *community structure* is the collection $\mathcal{C} = \{C_c\}$ of all communities. Notice that we explicitly allow a community to change or evolve over time. Once we have a community structure, we can derive from it an *affiliation sequence* for every individual x_i, the sequence $A_i = \langle \chi(v_{i,1}), \ldots, \chi(v_{i,t}) \rangle$ of communities that x_i was a member of during the observation period. Notice that it is possible that an individual has a color that does

not match any communities in $\mathcal{C}$. The interpretation is that the individual may be affiliated with an unknown community or not affiliated with any community at the time. Such individuals will not incur any absence costs but will incur visiting costs every time they are observed. This allows us to detect outliers whose affiliation cannot be clearly inferred from the observed interactions.

12.4.4 Discussion of Social Costs

The three event costs parametrize our method and are part of the input provided by the user based on knowledge of the social system under study. We can make observations about some general properties of these costs.

The actual numerical values of switching and visiting costs are irrelevant to the optimization. Only the *relative* values of these costs influence the optimal community interpretation. Altering the parameters $c_{\text{sw}}, c_{\text{vis}}, c_{\text{ab}}$ alters the dynamic expressiveness of the model. If c_{sw} is very large compared to c_{vis} and c_{ab} (such as a policy of death for betrayal), then in the optimal solution, individuals will never switch community affiliation, preferring to pay the visit and absence costs. At this extreme, we recover the static community structure as a special case. As c_{sw} gets smaller, more frequent affiliation changes are possible, and oscillating behavior between communities or gradual changes of community structure can be inferred as an explanation of the observations. If c_{sw} is very small compared to c_{vis} and c_{ab} (as perhaps caused by customer membership offers with switching incentives), then no visit or absence costs will be paid in the optimum, and individuals will switch every time they are in a group of a different community. As a result, we will obtain just the local community structures for individual time steps. Dynamic communities in this case are similar to those inferred by the "community tracing" methods of [5, 13, 14, 49]. Gradually changing the relative values of the costs interpolates between the local ephemeral structure and the global static view. It also allows us to infer the roles that individuals may play in their community. For example, individuals who rarely switch under any cost setting may be important to the persistence of their community. Among them those who rarely visit or are absent are potentially the core members of the community, while those who often are missing or visiting other communities are perhaps periphery members of the community. Those who switch very often under most cost settings are possibly the social butterflies. The choice of the three parameters determines the boundaries of these three categories. Thus, it is possible to systematically explore the entire range of relative cost values, in the process recovering the roles that individuals play and delineating the core and the periphery of communities.

Occasionally, it is possible to infer the real values of the costs explicitly. For example, many clubs and societies have membership and guest visit fees that place a monetary value on switches and visits. In some other cases, while it is impossible to precisely measure the value of switch, visit, and absence costs, it is possible to approximately measure their *relative* values. For instance, stress hormone profiles of

a newcomer to a group [41] or behavioral observation of aggression and harassment [40, 52] are fairly accurate stand-ins for costs.

Alternatively, one can view the costs of switching and visiting as probabilistic measures of individuals' behavior in a stochastic model that generates affiliation networks. A maximum likelihood set of probabilities can then be estimated from the fit of the resulting network to real data. In fact, our proposed dynamic community identification optimization framework can be viewed as exactly such a maximum likelihood estimator. The inferred probabilities of switching and visiting can help us understand which factors bring individuals together and tear them apart.

For the first formulation of the problem, we assumed for tractability that the costs are the same for all individuals and for all communities at all times. For a more realistic formulation of the DYNAMIC COMMUNITY IDENTIFICATION problem, this assumption must be relaxed; this is one of the many promising avenues of future research.

12.4.5 Complexity of the Problem

We investigate the computational complexity of the COMMUNITY INTERPRETATION problem which is an instantiation of the DYNAMIC COMMUNITY IDENTIFICATION optimization problem.

Unfortunately, solving the COMMUNITY INTERPRETATION problem optimally is NP-complete and, moreover, cannot be approximated arbitrarily well. We formally define the decision problem COMMUNITY INTERPRETATION as follows: *Given cost parameters* $c_{\text{sw}}, c_{\text{vis}}$*, and* c_{ab}*, a set* X *of* n *individuals and a sequence* $\mathcal{H} = \langle H_1, \ldots, H_T \rangle$ *of observations, as well as an upper bound* B *on the total cost, is there a community interpretation for* $(X, \mathcal{H})$ *of total cost at most* B*?*

Theorem 1 *The* COMMUNITY INTERPRETATION *problem is NP-complete and APX-hard.*[5]

Proof To prove APX-hardness, we give an approximation preserving reduction from the MINIMUM MULTIWAY CUT problem, which is known to be APX-hard [8]. We reduce from the special case of MINIMUM 3-WAY CUT (which remains APX-hard): Given an undirected graph $G = (V, E)$ with unit edge costs, and three distinct *terminal vertices* $s_1, s_2, s_3 \in V$ as well as a bound c, is there a set $C \subseteq E$ of at most c edges such that all of s_1, s_2, s_3 are disconnected from each other in the graph $(V, E \setminus C)$?

Given a 3-WAY CUT instance, we create a COMMUNITY INTERPRETATION instance. Let $n = |V|, m = |E|$. The COMMUNITY INTERPRETATION instance has an individual for each vertex $v \in V$. During each of the first $m + 1$ time steps, each of the singleton groups $\{s_1\}, \{s_2\}, \{s_3\}$ is observed (none of the other individuals

[5] The fact that the problem is APX-hard means that there is a constant $\varepsilon > 0$ such that, unless P=NP, no polynomial time algorithm can achieve an approximation guarantee better than $1 + \varepsilon$ for the problem.

are observed during those times). Let $e_1 = \{v_1, w_1\}, \ldots, e_m = \{v_m, w_m\}$ be an arbitrary ordering of the edges. During time step $m + 1 + t$, exactly one group $\{v_t, w_t\}$ is observed. That is, for each edge of G, the two endpoints are observed together exactly once. To complete the reduction, we set $c_{\text{sw}} = m + 1$, $c_{\text{vis}} = 1$, and $c_{\text{ab}} = 0$. To prove NP-completeness, we use the decision version of the COMMUNITY INTERPRETATION problem, asking whether there exists a solution of cost at most c. To prove that this is an approximation preserving reduction, we give a cost-preserving mapping from multiway cuts to valid colorings and vice versa. First, given a cut C, let S_1, S_2, S_3 be the connected components containing s_1, s_2, s_3, respectively. Give all vertices in S_i the same color in the coloring instance. For all vertices not in any of the three components (if any), arbitrarily color them with the same color as s_1. Finally, color each group of size 1 with the color of its unique member and each group of size 2 with the color of one of its members. First, notice that this is a valid coloring, as in the first $m + 1$ time steps, all singleton groups have distinct colors, and in the remaining m time steps, there is only one group in each time step. Because individuals never change color, no switching cost is incurred. Thus, the only cost is c_{vis} whenever an individual is present, but does not have the color of its group. This never happens for groups of one individual, and happens for groups of two individuals at time $m + 1 + t$ if and only if the corresponding edge e_t is cut. Thus, the total cost is exactly the same as the number of edges cut.

Conversely, if we have a valid coloring of the observation sequence, we notice that without loss of generality, it does not incur any switching cost, since a cheaper solution (of cost m) could always be obtained by simply assigning a fixed and distinct color to each individual. Second, we notice that each of the individuals s_1, s_2, s_3 must have distinct colors. Otherwise, they would incur a total visiting cost of at least $m + 1$ during the first $m + 1$ steps, and again, a cheaper solution could be obtained trivially by assigning all vertices a fixed distinct color. Define S_1, S_2, S_3 to be the sets of vertices with the same color as s_1, s_2, s_3, respectively. Let S_4 be the set of all remaining vertices (if any). Notice that we can assume, without the loss of generality, that all remaining vertices have the same color, otherwise a cheaper valid coloring could be obtained by coloring all of S_4 with the same color. Let C be the set of edges cut by the partition (S_1, S_2, S_3, S_4). By definition, C is a multiway cut separating s_1, s_2, s_3. Furthermore, the groups incurring a cost of $c_{\text{vis}} = 1$ in the coloring are exactly those corresponding to edges in C, as they are the ones with two distinct vertex colors, meaning that one of the vertices must have a color different from the group. (If any group of two vertices did not have the color of either of its individuals, we could obtain a cheaper solution by recoloring that group.) Thus, we have proved that the size of C is exactly equal to the cost of the group coloring we started with, completing the approximation preserving reduction. □

12.5 Algorithms for Finding Communities

Since the problem of inferring community structure in dynamic networks is NP-hard, for larger instances, we will use fast heuristics or approximation algorithms.

We present an approximation algorithm that produces a solution whose cost is within a constant factor of the optimum, regardless of the input size. It generates a group coloring first and then optimally colors the individual vertices. Recall that, for a minimization problem in general, a ρ-approximation algorithm is an algorithm which, on all instances of the problem, produces in polynomial time a solution whose cost is at most ρ times the cost of the optimal solution [57].

The idea behind the algorithm is to deal with one type of cost at a time. We first consider the special case where every individual is observed at all times and develop an algorithm based on maximum-weight bipartite matching for this simpler problem. Under this strong assumption, we prove that the algorithm is a ρ_1-approximation where $\rho_1 = \max\{1, 2c_{\text{sw}}/c_{\text{vis}}\}$. Hence, this algorithm always produces an optimal solution when $2c_{\text{sw}}/c_{\text{vis}} \leq 1$, that is, when one visit costs more than two switches.

Then, in Section 12.5.3, we consider the general problem where some individuals might at times be unobserved. For that problem, we present an algorithm based on the minimum-weight path cover problem and show that it is a ρ_2-approximation where $\rho_2 = \max\{2, 4c_{\text{sw}}/c_{\text{vis}}, 2c_{\text{sw}}/c_{\text{ab}}\}$.

The key observation for our algorithms is that once a coloring for the group vertices has been fixed, an optimum coloring for the individual vertices can be computed in polynomial time using dynamic programming.

At the heart of the dynamic programming approach is the observation that, given a fixed group coloring, the cost incurred by an individual does not depend on the colors chosen for other individuals.

Lemma 1 *Given a coloring of the group vertices, the minimum cost coloring of the individual vertices consists of minimum cost colorings of the vertices of each individual x_i, independent of other individuals.*

Proof The total cost is the sum of all switching, visiting, and absence costs over all relevant edges. The switching costs occur only for edges $(v_{i,t}, v_{i,t+1})$ and thus only depend on the colors of the corresponding individual x_i. Similarly, visiting and absence costs arise from present or absent edges $(v_{i,t}, u_{g,t})$ and thus only depend on the color of individual x_i, for a fixed group coloring. The total cost is the sum, over all individuals and all of their edges, of the corresponding switching, visiting, and absence costs and is thus minimized if the sum is minimized for all individuals x_i independently. □

In Section 12.5.4, we show how to find the optimal community membership sequence for one individual and, thus, by Lemma 1 for all individuals.

12.5.1 Group Graph

To find the group coloring, we use another auxiliary graph. The *group graph* of an interaction sequence $\mathcal{H} = \langle H_t \rangle$ is a directed acyclic graph $D = (V, E)$. The vertices of this graph are groups, and the edges show the intersection in membership

of the corresponding groups. This graph represents how the individuals *flow* from one group to another over time. The construction is quite similar (technically, it is the transitive reduction or Hasse diagram) to the metagroup graph in [5] or the group graph of [49]. Figure 12.4 shows an example group graph that corresponds to the interaction sequence and the cost graph in Fig. 12.3.

The nodes are the group nodes $u_{g,t}$ from our previous graph construction, as well as some dummy group vertices we describe below. (There are no individual vertices.) For each pair of vertices$u_{g,t}, u_{g',t'}$ with $t' > t$, we have a directed

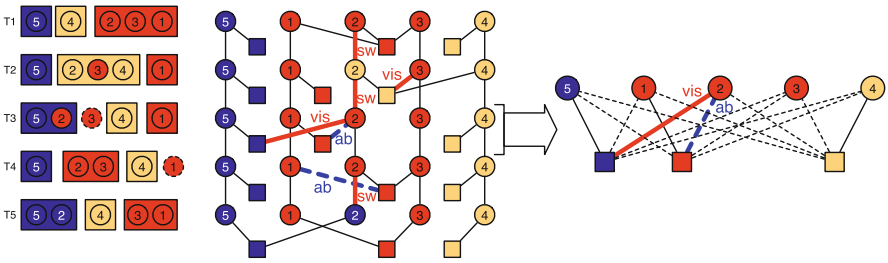


Fig. 12.3 An example of an interaction sequence (*left*) and the corresponding graph representation (*middle and right*). In the interaction sequence, each row corresponds to a time step, with time going from top to bottom. Each *rectangle* represents an observed group, and the *circles* within are the member individuals. The circles without an enclosing rectangle are the unobserved individuals. Similarly, in the graph representation, the squares are group vertices and the circles are individual vertices. Not all edges or incurred costs are drawn in the middle graph for visibility. The graph on the right shows the complete representation of time step T3. The E_{vis} edges are shown as *solid lines* and the E_{ab} edges as *dashed lines*. Since individual 3 is not observed in time step T3, it only has E_{ab} type edges in this time step. The coloring shown is one example community interpretation. Some of the $c_{\text{sw}}, c_{\text{vis}}, c_{\text{ab}}$ costs under this coloring are shown in the graph representations in the middle and right

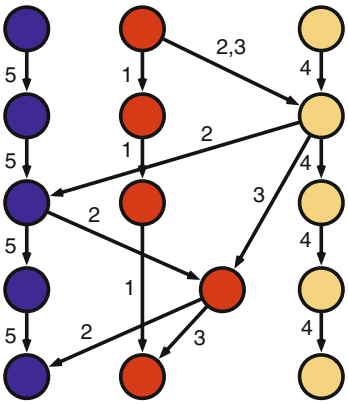


Fig. 12.4 The group graph that corresponds to the interaction sequence and the cost graph in Fig. 12.3. The edges are labeled with the individuals in $\lambda(g, h)$. There are no dummy vertices since every individual is observed at the first and last time steps

edge $(u_{g,t}, u_{g',t'})$ with a label $\lambda(u_{g,t}, u_{g',t'})$ and weight $w(u_{g,t}, u_{g',t'})$. The label is the set of all individuals who are in both g and g' at the respective times and were not observed in any group at any time between t and t'. Formally, $\lambda(u_{g,t}, u_{g',t'}) = \left\{x_i \in g \cap g' : x_i \notin g_{j,t''}, \text{ for all } g_{j,t''} \in H_{t''}, \text{ for all } t'' \in (t, t')\right\}$. The weight is simply the number of individuals in the label: $w(u_{g,t}, u_{g',t'}) = |\lambda(u_{g,t}, u_{g',t'})|$. We remove all edges whose labels are empty. If all individuals are observed in each time step, then edges only go from vertices $u_{g,t}$ to vertices $u_{g',t+1}$. That is, edges never skip time steps.

The dummy group vertices account for the individuals who were not observed in the first or last time step. Specifically, for any group $g_{j,t} \in H_t$, let $g^-_{j,t} \subseteq g_{j,t}$ be the set of all individuals in $g_{j,t}$ who are not in any group $g_{j',t'} \in H_{t'}$ for any $t' < t$. Similarly, $g^+_{j,t} \subseteq g_{j,t}$ denotes the set of all individuals in $g_{j,t}$ who are not in any group $g_{j',t'} \in H_{t'}$ for $t' > t$. Whenever $g^-_{j,t} \neq \emptyset$, we add a dummy vertex $u_{g^-_{j,t},1}$ at time step 1. Similarly, when $g^+_{j,t} \neq \emptyset$, we add a dummy vertex $u_{g^+_{j,t},T}$ at time step T. The edges, labels, and weights from or to the dummy vertices are constructed as for regular vertices. Of course, if all individuals are observed in each time step, then no dummy vertices are constructed.

Algorithm 1 shows how to construct the group graph. The running time of Algorithm 1 is $\Theta\left(T\left(\sum_{t=1}^{T} |H_t|\right)^2\right)$. This running time cannot be improved since it is possible that every pair of groups in different time steps have at least one member in common (i.e., $w(u_{g,t}, u_{g',t'}) > 0$ for every $g \in H_t$, $g' \in H_{t'}$, and $t < t'$).

Algorithm 1 CREATEGROUPGRAPH

Require: interaction sequence $\mathcal{H} = \langle H_t \rangle$.
Add dummy vertices to H_1 and H_T.
for $t = 1, \ldots, T-1$ **do**
  for $g \in H_t$ **do**
    $A \leftarrow g$
    for $t' = t+1, \ldots, T$ **do**
      for $g' \in H_{t'}$ **do**
        if $g' \cap A \neq \emptyset$ **then**
          $E \leftarrow E \cup \left\{(u_{g,t}, u_{g',t'})\right\}$
          $\lambda(u_{g,t}, u_{g',t'}) \leftarrow A \cap g'$
          $w(u_{g,t}, u_{g',t'}) \leftarrow |A \cap g'|$
          $A \leftarrow A \setminus g'$
return $D = (V, E)$

12.5.2 Approximation via Bipartite Matching

In this section, we assume that all individuals are observed at all time steps. The algorithm, first presented in [54] without analysis, is based on finding a maximum-weight matching on bipartite graphs, also known as the assignment

problem. (The approach is somewhat similar to [49].) The matching problem can be solved efficiently [29]. Once we have the matching on every bipartite subgraph G_t, the subgraph D' induced by the set of matched edges consists of vertex-disjoint paths (since the edges of D' are matched edges). For each path, coloring all groups on that path and all individuals in these groups with the same color clearly takes linear time. Thus, the overall time of the Algorithm MATCHING COMMUNITIES is bounded by the time of finding the maximum-weight matchings on the bipartite graphs. The pseudocode for the algorithm is shown as Algorithm 2.

Algorithm 2 MATCHINGCOMMUNITIES (MC)

Require: An interaction sequence $\mathcal{H}$.
1: $D = (V, E) \leftarrow$ undirected version of the group graph of $\mathcal{H}$, dropping edge orientations.
2: **for** time $t = 1, \ldots, T-1$ **do**
3: $G_t \leftarrow$ bipartite subgraph of D induced by $\bigcup_{g \in H_t} u_{g,t} \cup \bigcup_{g \in H_{t+1}} u_{g,t+1}$, the sets of group vertices at times t and $t+1$.
4: $M_t^* \leftarrow$ maximum weight matching on G_t.
5: $D' = (V, \cup_{t=1}^{T-1} M_t^*) \leftarrow$ the group graph D with the edge set replaced by the matched edges.
6: Color (the groups in) each connected component of D' by a distinct color.
7: Color each individual at each time step by the same color as the group in which it was observed so that all groups are monochromatic.

Theorem 2 *For convenience, let* $\mu_1 = \min\left\{c_{\text{sw}}, \frac{c_{\text{vis}}}{2}\right\}$, $\rho_1 = \frac{c_{\text{sw}}}{\mu_1} = \max\left\{1, \frac{2c_{\text{sw}}}{c_{\text{vis}}}\right\}$. *Given an interaction sequence* $\mathcal{H}$ *with all individuals present at all times, Algorithm* MC *produces, in polynomial time, a coloring with cost at most* ρ_1 *times that of the optimum.*

The proof is technical and we refer the reader to [53, Theorem 3] for details.

When $2c_{\text{sw}} \leq c_{\text{vis}}$ then $\rho_1 = 1$ and the algorithm always produces an optimal coloring.

12.5.3 Approximation via Path Cover

We now relax the assumption that all individuals are observed in all time steps. We present a ρ_2-approximation algorithm for the general case and analyze its performance guarantee. Before describing the algorithm, we recall the definition of a path cover of a graph.

12.5.3.1 Path Cover Problem

In a directed graph $D = (V, E)$, a *directed path* is a sequence of distinct vertices $P = v_1, \ldots, v_k$ such that (v_i, v_{i+1}) is an edge of D for every $i = 1, 2, \ldots, k-1$. Two directed paths P_1 and P_2 are *vertex-disjoint* if they share no vertices. A *path cover* $\mathcal{P}$ on D is a set of pairwise vertex-disjoint paths [11] in which every vertex

lies on (is covered by) *exactly* one path in $\mathcal{P}$. The MINIMUM PATH COVER problem is to find a path cover with the minimum number of paths. The decision version of the problem on general graphs is NP-complete [19]. However, on directed acyclic graphs (DAGs), the problem can be solved in polynomial time via a reduction to the matching problem in bipartite graphs [7].

12.5.3.2 Algorithm Description

The approximation algorithm works as follows: We reserve one *absence color* ε not to be assigned to any group. Intuitively, individuals with color ε are considered to be in the "missing" community at the time.

The algorithm runs in polynomial time. Furthermore, a coloring χ produced by PCC is valid, because groups of the same color c_P must lie on a common path P, and all edges go strictly forward in time.

Algorithm 3 PATHCOVERCOMMUNITIES (PCC)

Require: An interaction sequence $\mathcal{H}$.
1: $D = (V, E) \leftarrow$CREATEGROUPGRAPH($\mathcal{H}$)
2: $\mathcal{P}^* \leftarrow$ minimum weight path cover on D.
3: **for all** paths $P \in \mathcal{P}^*$ **do**
4: Color all real groups g with $u_{g,t} \in P$ with the same color c_P (specific to path P).
5: **for all** edges $e = (u_{g,t}, u_{g',t'}) \in P$ **do**
6: **for all** times $t'' = t, \ldots, t'$ **do**
7: Color each individual $x_i \in \lambda(e)$ at time t'' with the color c_P.
8: Color the remaining vertices by ε.

Theorem 3 *Algorithm* PCC *is a* ρ_2*-approximation where*

$$\rho_2 = 2 \cdot \frac{c_{\text{sw}}}{\mu_2} = 2 \cdot \max\left\{1, \frac{2c_{\text{sw}}}{c_{\text{vis}}}, \frac{c_{\text{sw}}}{c_{\text{ab}}}\right\}, \quad \mu_2 = \min\left\{c_{\text{sw}}, \frac{c_{\text{vis}}}{2}, c_{\text{ab}}\right\}.$$

We refer the reader to [53, Theorem 6] for the proof of the theorem.

12.5.4 Optimal Individual Coloring

Although PCC has a provably good approximation guarantee, in practice, we can easily improve the individual coloring further by optimally coloring individuals after the group coloring is obtained. We use dynamic programming to obtain this coloring.

Recall that by Lemma 1, it is sufficient to describe an algorithm for optimally coloring any one individual over the T time steps since its coloring is independent

of that for other individuals. By running this algorithm for each individual x_i, we obtain a complete optimal coloring. Let x_i be one individual.

Let $\chi_g(t)$ be the color of the group in which x_i participates at time step t, $\chi_g(t) = \varepsilon$ if x_i was unobserved. By this convention, an individual with color ε at time step t is said to be *unaffiliated* with any communities at that time step. Let $C(i) = \{\varepsilon\} \cup \left\{\chi_g(t) : t = 1, \ldots, T\right\}$ be the set of all group colors of x_i, including the absence color ε. There is an optimal coloring for x_i which uses colors in $C(i)$; any other color would incur more cost without any compensating benefit.

Let $c_{\text{sw}}(t, p, q)$ be the switching cost when coloring i at time steps t and $t-1$ with colors p and q, respectively. Let $c_{\text{vis}}(t, p)$ and $c_{\text{ab}}(t, p)$ be the visiting cost and absence cost, respectively, when coloring i at time step t with color p. Notice that $c_{\text{sw}}(t, p, q)$, $c_{\text{vis}}(t, p)$, and $c_{\text{ab}}(t, p)$ can all be easily computed given the group coloring and the parameters c_{sw}, c_{vis}, and c_{ab}. The following recurrence computes the minimum cost of coloring i in time step t with color p:

$$\begin{aligned} c(t, p) &= \min_{q \in C(i)} \{c(t-1, q) + c_{\text{sw}}(t, p, q) + c_{\text{vis}}(t, p) + c_{\text{ab}}(t, p)\}, \quad t \geq 2 \\ c(1, p) &= c_{\text{vis}}(1, p) + c_{\text{ab}}(1, p). \end{aligned} \tag{12.1}$$

Since the dynamic program explicitly optimizes over all relevant choices for time steps t and $t-1$, we obtain the following Lemma by induction.

Lemma 2 *Given a group coloring, the optimal cost of coloring an individual x_i at time step t with a color $p \in C(i)$ is given by Recurrence* (12.1). □

Applying this lemma to the end of the time horizon T, we derive the following theorem:

Theorem 4 *Given a group coloring, the minimum cost of coloring the individual x_i is* $\min_{p \in C(i)} c(T, p)$. □

The dynamic programming approach can be implemented efficiently by noting that only the optimal solution for the immediately preceding time step needs to be retained at any point for computing the solution at time t. Thus, the table at step t is of size $O(C)$ where C is the total number of colors. We also record which coloring gave rise to the optimal solution in each time step. Finding the optimum value for a pair (t, p) involves trying all colors $q \in C$. Thus, each of the $O(TC)$ entries is computed in time $O(C)$, for a total running time of $O(TC^2)$. Finally, the dynamic program is run independently for each of the individuals i, giving a total running time of $O(nTC^2)$, with a space requirement of $O(TC)$.

The optimal individual coloring algorithm can be applied to any group coloring, obtained by any method, including those that track community evolution [1, 5, 13, 14, 37, 49]. However, the resulting individual coloring is optimal only under the model of social costs introduced here.

12.6 Experimental Validation

We first show that the COMMUNITY INTERPRETATION optimization problem produces meaningful communities by examining the optimal solution (found by exhaustive search). We then show that the proposed approximation algorithm results in communities similar to the optimum and thus performs well in practice. Since the COMMUNITY INTERPRETATION problem is NP-hard, we can only find the optimal solution for small data sets. Thus, we first validate our approach on small synthetic and real data sets and also use those small data sets to compare the approximation algorithm to the optimal solution. Once both the definition and the algorithm are validated, we proceed to apply them to larger practical data sets.

We begin by inferring communities in a synthetic data set with known embedded communities. Then, we infer communities in a well-studied benchmark data set.

12.6.1 Theseus' Ship

The *Theseus' Ship* example models communities in which small changes happen in each time step, aggregating to a complete change in membership over longer periods of time. Real-world examples of such communities include sports teams, students in a department, lab, or school, casts of recurring TV shows, or cells in the human body [48] (as well as the individual parts of the Ship of Theseus [44]). Recall that a community is the identity of its members. Yet as the membership of a community changes, and even completely turns over, the question then arises, "What happens with that identity?" Does it go with the majority of its members or does it stay with the independent concept of a community? This philosophical question, known as the Theseus' Ship Paradox [44], has occupied philosophers from ancient Greece to Leibnitz. We reconcile both sides of the debate by varying the input social costs.

A generalized Theseus' Ship has $n = km$ individuals and m groups. In time step t, the ith group consists of individuals $(ki+t) \mod n, \ldots, (ki+t+k-1) \mod n$, that is, in each time step, the lowest-numbered member of each group moves to the next lower group (wrapping around at n). Figure 12.5 shows an example of a classic Theseus' Ship with $n = 18$ individuals and $m = 2$ groups.

Figure 12.5a shows the optimal coloring under the cost setting $(c_{\text{sw}}, c_{\text{vis}}, c_{\text{ab}}) = (4, 1, 0)$ in which the switching cost is relatively high. Thus, individuals do not change colors, and the identity of the community follows the majority of its members. On the other hand, with the cost setting $(c_{\text{sw}}, c_{\text{vis}}, c_{\text{ab}}) = (1, 2, 0)$, the visiting cost is high. Figure 12.5b shows that the resulting coloring has individuals change their community membership to match their group. Thus, the community identity of groups stays the same even as the individual members change. In the aggregate view (Fig. 12.6) only a single community can be inferred.

12.6.2 Southern Women

Southern Women [10] is a data set collected in 1933 in Natchez, TN, by a group of anthropologists conducting interviews and observations over a period of 9 months. It

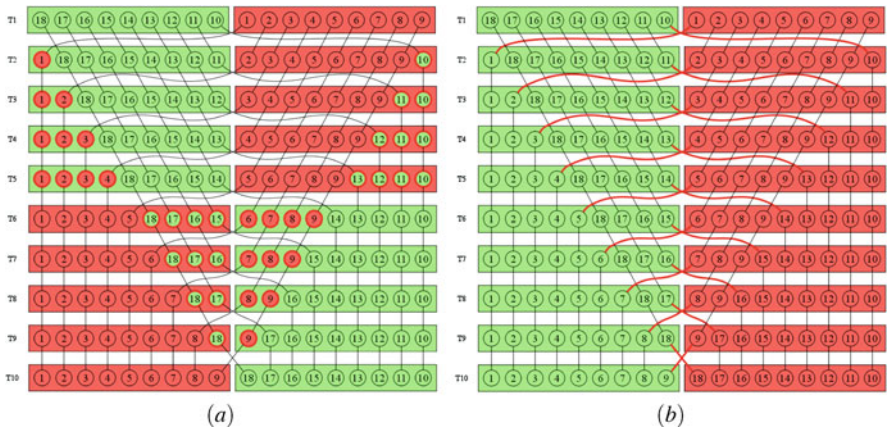


Fig. 12.5 Optimal colorings of the Theseus' Ship example with costs $(c_{sw}, c_{vis}, c_{ab}) = (4, 1, 0)$ and $(1, 2, 0)$

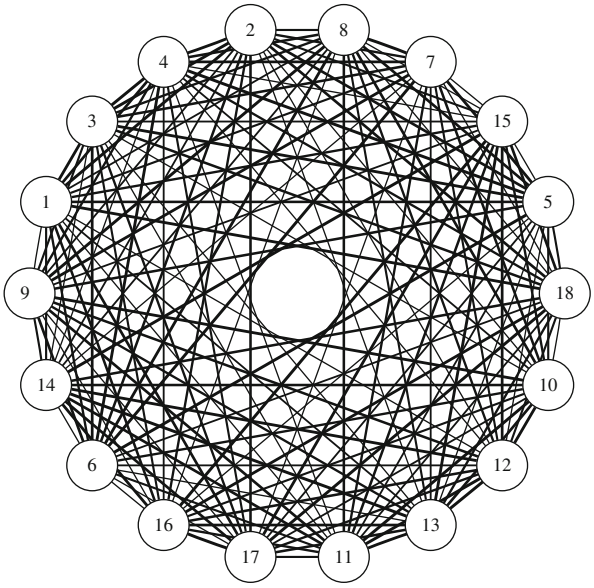


Fig. 12.6 Aggregate (static) view of the Theseus' Ship network

tracks 18 women and their participation in 14 informal social events such as garden parties and card games. The data set has been extensively studied and used as a benchmark for community identification methods [21]. The event participation table is shown in Table 12.1, taken verbatim from [10]. The columns, each representing

an event, are not ordered chronologically, but are manually arranged by the table authors to illustrate two communities at the upper-left and lower-right corners.

A summary of the community identification results of 21 methods was performed by Freeman [21] and is shown in Table 12.2.

In applying our method on this data set, since there is only one group being present in each time step of the data set, it is essential that we use the absence cost $c_{ab} \neq 0$, otherwise the optimal solution would trivially be obtained by coloring all groups and individuals with the same color. We consider the cost setting $c_{sw} = c_{vis} = c_{ab} = 1$. Figure 12.7a–c shows the colorings by our methods: (a) PCC alone, (b) PCC with dynamic programming, and (c) an optimal coloring by exhaustive search.

The structure of communities found by PCC and by the exhaustive search is fairly similar. Many of the groups are already the same and PCC with dynamic programming captures much of the visiting and absence information present in the optimal solution. The application of dynamic programming decreases the cost by approximately 40%. Dynamic programming, within the limits of the initial PCC solution, takes the costs into consideration in the best possible way to improve the coloring for individuals.

To compare our dynamic communities to the results of existing static community detection algorithms, we need to aggregate them meaningfully over time. One way to infer a static community structure from the dynamic one is to assign to an individual the color of the community with which it is affiliated the majority of the time. In Fig. 12.7 individuals 1–4 belong to the blue community since they were observed in blue groups while being colored blue 4–5 times each, while individuals 11–15 belong to the yellow community by a similar argument. Individuals 5–7 were in blue groups while being colored blue 1–3 times and were in groups of other colors at most once, making them members of the blue community, though their ties to the community were not as strong as those of individuals 1–4. The rest of the population were rarely observed. Thus, their affiliations cannot be clearly inferred from the data. Comparing to the results of the 21 static methods in Table 12.2, our interpretation of communities is consistent with almost all of the static methods (except method number 20 [OSB00] which identifies individuals 1–16 to be in one big community). We do not delve into the details of the dynamic community structure for Southern Women, since with very few observations and one group per observation, little interesting dynamic information can be gleaned. Rather, we use this data set as a benchmark to show that our approach recovers meaningful communities that align with intuitive and analytical notions of a community for a well-studied data set.

12.7 Applications to Real-World Data Sets

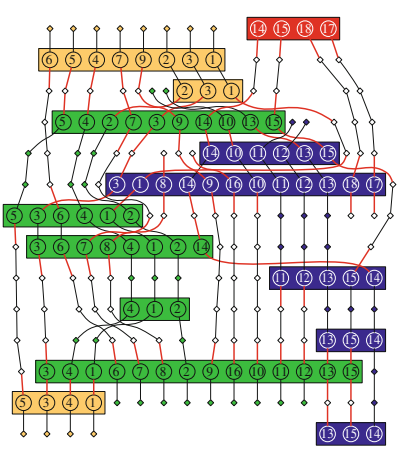
To test the scalability of the proposed approximation algorithm, we use several real-world interaction data sets of various sizes as shown in Table 12.3.

Table 12.1 Reproduction of the Southern Women data set [21]

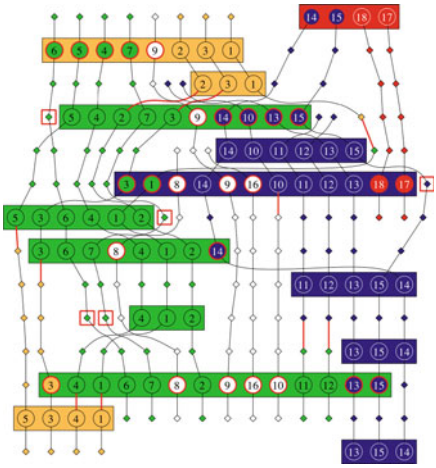
	Social events in *Old City Herald*													
Names of participants	6/27	3/2	4/12	9/26	2/25	5/19	3/15	9/16	4/8	6/10	2/23	4/7	11/21	8/3
1. Mrs. Evelyn Jefferson	x	x	x	x	x	x		x	x					
2. Miss Laura Mandeville	x	x	x		x	x	x	x						
3. Miss Theresa Anderson		x	x	x	x	x	x	x	x					
4. Miss Brenda Rogers	x		x	x	x	x	x	x						
5. Miss Charlotte McDowd			x	x	x		x							
6. Miss Frances Anderson			x		x	x		x						
7. Miss Eleanor Nye					x	x	x	x						
8. Miss Pearl Oglethorpe						x		x	x					
9. Miss Ruth DeSand					x		x	x	x					
10. Miss Verne Sanderson							x	x	x			x		
11. Miss Myra Liddell								x	x	x		x		
12. Miss Katherine Rogers								x	x	x		x	x	x
13. Mrs. Sylvia Avondale							x	x	x	x		x	x	x
14. Mrs. Nora Fayette						x	x		x	x	x	x	x	x
15. Mrs. Helen Lloyd							x	x		x	x	x		
16. Mrs. Dorothy Murchison								x	x					
17. Mrs. Olivia Carleton									x		x			
18. Mrs. Flora Price									x		x			

Table 12.2 Assignments of individuals in Southern Women to communities A, B, and C by the 21 static methods, reproduced from [21]

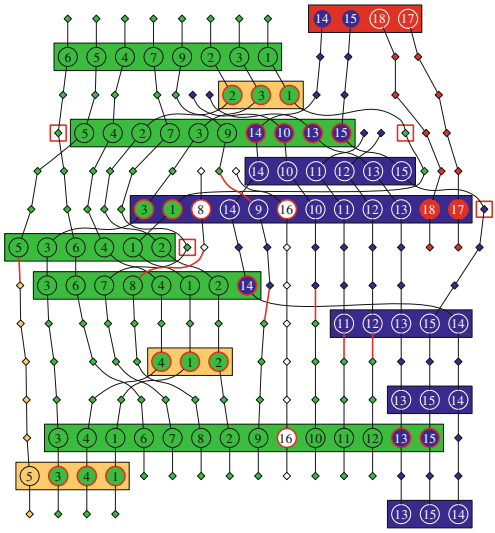
Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1. DGG41	A	A	A	A	A	A	A	A	AB	B	B	B	B	B	B	B	B	B
2. HOM50	A	A	A	A	A	A	A	AB			B	B	B	B	B		B	B
3. P&c72	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
4. BGR74	A	A	A	A	A	A	A		A	B	B	B	B	BC	BC		C	C
5. BBA75	A	A	A	A	A	A	A	B	A	B	B	B	B	B	B	B	B	B
6. BCH78	A	A	A	A	A	A				B	B	B	B	B	B			
7. DOR79	A	A	A	A	A	A	A		A	B	B	B	B	B	B			
8. BCH91	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
9. FRE92	A	A	A	A	A	A	A		A	B	B	B	B	B	B	B		
10. E&B93	A	A	A	A	A	A	A		A	B	B	B	B	B	B			
11. FR193	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
12. FR293	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B	B	B
13. FW193	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	AB	B	B
14. FW293	A	A	A	A	A	A	A		A	B	B	B	B	B	B		B	B
15. BE197	A	A	A	A	A	A	A		A	B	B	B	B	B	B			
16. BE297	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
17. BE397	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
18. S&F99	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B		B	B
19. ROB00	A	A	A	A	A	A	A	A	A	B	B	B	B	B	B	B	B	B
20. OSB00	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	B	B
21. NEW01	A	A	A	A	A	A	A	B	A	B	B	B	B	B	B	B	B	B



(a) PCC solution, cost 74



(b) PCC with dynamic programming solution, cost 43



(c) Optimal solution, cost 36

Fig. 12.7 Dynamic communities inferred by PCC alone, PCC with dynamic programming, and the optimal solution on the Southern Women data set with input costs $c_{\text{sw}} = c_{\text{vis}} = c_{\text{ab}} = 1$

12.7.1 Data Sets

Our data sets fall into two categories: interactions among animals (equids, wild zebras, and onagers) and among humans carrying mobile devices.

12.7.1.1 Animal Interactions

We consider three data sets capturing affiliations among members of three species: equids, wild zebra, and onagers (wild asses). All three were obtained in similar

Table 12.3 Statistics of the data sets used in testing. n is the number of individuals, T the number of time steps, and "groups" shows the number of groups for each data set

Data set	n	T	Groups
Grevy's zebra	27	44	75
Onagers	29	82	308
Plains zebra	2510	1268	7907
Haggle-41	41	418	2131
Haggle-264	264	425	1411
Reality mining	96	1577	3958

ways, by observing spatial proximity of members of a population over a period of time. Predetermined census loops were driven approximately 5 times per week. Individual animals were identified by their unique markings such as stripe patterns of zebras or scars and ear notches of onagers [51].

- The Grevy's zebra (*Equus grevyi*) data set was collected by observing the population over 3 months in 2002 in Kenya [51]. Zebras are uniquely identifiable by the pattern of stripes on various parts of their bodies. The data were collected by ecologists making visual scans of the herds, typically once a day. Each entity in the dynamic network is a unique Grevy's zebra, and an interaction represents social association, as determined by spatial proximity and the domain knowledge of ecologists.
- The onagers (*Equus hemionus khur*) data set was collected by observing a population of onagers (wild asses) in the Little Rann of Kutch desert in Gujarat, India [51], from January to May in 2003. Individual onagers were identified by markings on their body and, similar to zebras, the data represent visual scans of the population by ecologists, typically once a day. An interaction represents an association as determined by physical proximity and domain knowledge.
- The Plains Zebra (*Equus burchelli*) data set was collected by observing the population in Kenya from 2003 until 2008 [16, 28, 51], in a manner similar to the Grevy's zebra data set.

12.7.1.2 Human Interactions

In addition to the animal interaction data sets, we use three networks derived from interactions of humans carrying Bluetooth devices.

- Reality mining is one of the largest mobile phone projects attempted in academia. The data were collected by the MIT Media Lab [12]. They consist of communication, proximity, location, and activity information from 100 subjects at MIT over the course of the 2004–2005 academic year.
- The Haggle data set consists of social interactions among attendees of the 2005 IEEE Infocom conference, carrying a Bluetooth-enabled device for

recording proximity over time [43]. Two separate data sets represent interactions at this event. One consists only of 41 participants and the other consists of 264 nodes, containing the 41 participants and the other devices in proximity. The participants were tracked over the full 4 days of the conference at 10 min intervals.

12.7.2 Results

On each data set, we infer the dynamic community structure under several cost settings, by PCC alone and PCC with dynamic programming. The algorithms take only a few seconds on most data sets, up to a few minutes on the largest one; we therefore omit the precise times here. We fix $c_{\text{ab}} = 1$ for simplicity and compare the cost of the communities inferred by the algorithms to the theoretical upper bound on the worst-case performance ratio of the PCC algorithm. Most of the data sets are too large for an exhaustive search. Therefore, we use the inverse of the worst-case performance ratio as the lower bound on the optimum and, hence, on the algorithms' solutions.

Figure 12.8 shows the resulting costs for the dynamic community structures inferred by the various methods as well as the theoretical bounds.

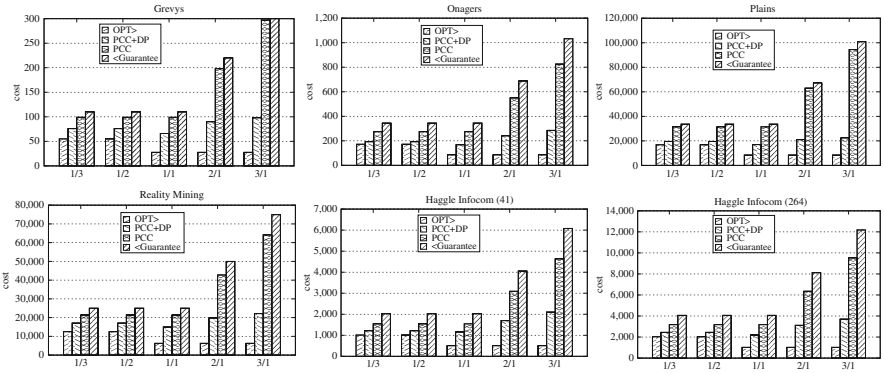


Fig. 12.8 Performance of the algorithms PCC alone and PCC with dynamic programming, compared to a bound on optimal solution on various data sets

The performance ratios of the algorithms increase as the ratio $c_{\text{sw}}/c_{\text{vis}}$ increases. Intuitively, the more expensive the switching cost, the worse the two algorithms perform, since PCC does not take into account the ratio between c_{sw} and c_{vis}. However, dynamic programming compensates for this shortcoming of PCC considerably in practice. In the worst case of PCC, when $c_{\text{sw}} = 3c_{\text{vis}}$, the dynamic programming reduces the cost by approximately 63–77%. The most dramatic difference between the two algorithms can be seen on the Plains data set. On this data set, PCC performs

the worst, producing a coloring with cost 11.44 times the bound on the optimal solution; dynamic programming reduces the cost to 2.62 times the bound on the optimal solution.

Overall, remarkably, the PCC algorithm with dynamic programming finds a community structure of social cost reasonably close to the bound on the optimum. Moreover, where we could obtain the actual optimal solution, the community structure found by the algorithm is qualitatively similar to that of the optimum.

Unfortunately, at the moment, there are no metrics for comparing two dynamic community structures, or two dynamic clusterings, for that matter. Thus, beyond the social cost and evaluation by domain experts, we have no basis to compare two different structures or two different algorithms.

12.8 Conclusions

The problem of community identification in dynamic networks or, more generally, dynamic clustering, is in the very early stages of being studied. In this chapter, we have outlined the three major approaches of tackling the problem: (1) traditional clustering within each time step and matching those clusters across time steps [1, 5, 14], (2) maximizing the likelihood of the inferred community structure under some social model [32, 59], and (3) directly considering dynamic communities as clusters over time [50, 53, 54].

Our method, outlined here in detail, is at the moment the only one that subsumes all three of the above approaches. It views dynamic communities as dynamic clusters under the distance measure of some notion of a social cost. Defined as a combinatorial optimization problem that finds a community structure that minimizes the overall social cost, it turns out to be equivalent to a maximum likelihood community structure for a very simple Markovian model of social group behavior. Finally, the idea of matching across time steps clusters from within each time step is used to design a powerful, fast, and accurate approximation algorithm.

Although our approach is capable of inferring plausible and useful community structures in diverse dynamic social network data sets, the chapter on dynamic community inference is by no means closed. None of the existing methods can take a large arbitrary dynamic social network and infer community structure (if one exists) at arbitrary temporal scales. Moreover, at the moment, there are no benchmark data sets (although some are emerging) or an agreed-upon way to generate synthetic test data for dynamic community inference. There is no consensus on even the most basic way to compare dynamic community structures to evaluate relative performance of various methods or to compare them to ground truth if one exists. Thus, we hope that this chapter will serve as a starting point and as motivation for future research on dynamic community inference which at the moment is quite an open problem.

References

1. C. C. Aggarwal and P. S. Yu. Online analysis of community evolution in data streams. In *Proceedings of the 5th SIAM International Conference on Data Mining*, pages 56–67, Philadelphia, PA, USA, 2005. SIAM.
2. L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pages 44–54, Philadelphia, PA, USA, 2006. ACM, New York, NY.
3. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.
4. P. S. Bearman, J. Moody, and K. Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. *American Journal of Sociology*, 110(1):44–91, July 2004.
5. T. Y. Berger-Wolf and J. Saia. A framework for analysis of dynamic social networks. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pages 523–528, Philadelphia, PA, USA, 2006. ACM Press, New York, NY.
6. R. L. Breiger. The duality of persons and groups. *Social Forces*, 53(2):181–190, December 1974.
7. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, 2001.
8. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.
9. L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, September 2005.
10. A. Davis, B. B. Gardner, and M. R. Gardner. *Deep South*. The University of Chicago Press, Chicago, IL, 1941.
11. R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, Heidelberg, August 2005.
12. N. Eagle and A. (Sandy) Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, May 2006.
13. T. Falkowski. *Community Analysis in Dynamic Social Networks*. Dissertation, University Magdeburg, 2009.
14. T. Falkowski, J. Bartelheimer, and M. Spiliopoulou. Mining and visualizing the evolution of subgroups in social networks. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 52–58, Hong Kong. IEEE Computer Society, 2006.
15. K. Faust. Scaling and statistical models for affiliation networks: patterns of participation among Soviet politicians during the Brezhnev era. *Social Networks*, 24(3):231–259, July 2002.
16. I. R. Fischhoff, S. R. Sundaresan, J. Cordingley, and D. I. Rubenstein. Habitat use and movements of plains zebra (*Equus burchelli*) in response to predation danger from lions. *Behavioral Ecology*, 18(4):725–729, June 2007.
17. S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, February 2010.
18. S. Fortunato and C. Castellano. Community structure in graphs. *eprint arXiv: 0712.2716*, 2007.
19. D. S. Franzblau and A. Raychaudhuri. Optimal Hamiltonian completions and path covers for trees, and a reduction to maximum flow. *The Australian & New Zealand Industrial and Applied Mathematics Journal*, 44(2):193–204, October 2002.
20. L. C. Freeman. On the sociological concept of "group": a empirical test of two models. *American Journal of Sociology*, 98(1):152–166, July 1993.
21. L. C. Freeman. Finding social groups: A meta-analysis of the southern women data. In R. Breiger, K. Carley, and P. Pattison, editors, *Dynamic Social Network Modeling and Analysis*, pages 39–98. The National Academies Press, Washington, DC, 2003.
22. W. Fu, L. Song, and E. P. Xing. Dynamic mixed membership blockmodel for evolving networks. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 329–336, Montreal, Canada, 2009. ACM, New York, NY.

23. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Science USA*, 99(12):7821–7826, June 2002.
24. S. Hanneke and E. Xing. Discrete temporal models of social networks. In *Statistical Network Analysis: Models, Issues, and New Directions, Proceedings of the ICML 2006 Workshop on Statistical Network Analysis*, volume 4503 of *Lecture Notes in Computer Science*, pages 115–125, 2007. Springer, Heidelberg, German.
25. J. Hopcroft, O. Khan, B. Kulis, and B. Selman. Natural communities in large linked networks. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining*, pages 541–546, Washington, DC, USA, 2003. ACM, New York, NY.
26. A. Iriberri and G. Leroy. A life-cycle perspective on online community success. *ACM Computing Survey*, 41(2):1–29, February 2009.
27. T. R. Jensen and B. Toft. *Graph Coloring Problems*. Discrete Mathematics and Optimization. Wiley-Interscience, New York, NY, 1994.
28. P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. I. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. *ACM SIGPLAN Notices*, 37(10):96–107, October 2002.
29. J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2005.
30. R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. In *Proceedings of the 12th International Conference on World Wide Web*, pages 568–576, Budapest, Hungary, 2003. ACM, New York, NY.
31. S. Lattanzi and D. Sivakumar. Affiliation networks. In *Proceedings of the 40th ACM Symposium on Theory of Computing*, pages 427–434, Bethesda, MD, USA, 2009. ACM, New York, NY.
32. Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng. FacetNet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th International Conference on World Wide Web*, pages 685–694, Beijing, China, 2008. ACM, New York, NY.
33. D. Lusseau, H. Whitehead, and S. Gero. Incorporating uncertainty into the study of animal social networks. *Animal Behaviour*, 75(5):1809–1815, May 2008.
34. M. McPherson, L. S. Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, January 2001.
35. M. Newman, A.-L. Barabási, and D. J. Watts, editors. *The Structure and Dynamics of Networks*. Princeton University Press, Princeton, NJ, 2006.
36. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, February 2004.
37. G. Palla, A.-L. Barabási, and T. Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664–667, April 2007.
38. M. Pearson and P. West. Drifting smoke rings: Social network analysis and Markov processes in a longitudinal study of friendship groups and risk-taking. *Connections*, 25(2):59–76, 2003.
39. M. A. Porter, J.-P. Onnela, and P. J. Mucha. Communities in networks. *Notices of the American Mathematical Society*, 56(9):1082–1097, 1164–1166, October 2009.
40. D. I. Rubenstein and C. M. Nuñez. *Sociality and reproductive skew in horses and zebras*, pages 196–226. Reproductive Skew in Vertebrates: Proximate and Ultimate Causes. Cambridge University Press, Cambridge, U.K., 2009.
41. R. M. Sapolsky. The endocrine stress-response and social status in the wild baboon. *Hormones and behavior*, 16(3):279–292, September 1982.
42. P. Sarkar and A. Moore. Dynamic social network analysis using latent space models. *ACM SIGKDD Explorations Newsletter*, 7(2):31–40, December 2005.
43. J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau. CRAWDAD trace cambridge/haggle/imote/infocom (v. 2006-01-31). Downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/infocom, January 2006.
44. D. Sedley. The Stoic criterion of identity. *Phronesis*, 27:255–275, 1982.

45. T. Snijders, C. Steglich, and G. van de Bunt. Introduction to actor-based models for network dynamics. *Social Networks*, 32(1):44–60, January 2009.
46. T. A. Snijders. Models for longitudinal network data. In P. Carrington, J. Scott, and S. Wasserman, editors, *Models and methods in social network analysis*, chapter 11. Cambridge University Press, New York, 2005.
47. T. A. B. Snijders. The statistical evaluation of social network dynamics. *Sociological Methodology*, 31:361–395, 2001.
48. K. L. Spalding, R. D. Bhardwaj, B. A. Buchholz, H. Druid, and J. Frisén. Retrospective birth dating of cells in humans. *Cell*, 122(1):133–143, 2005.
49. M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. MONIC: modeling and monitoring cluster transitions. In *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining*, pages 706–711, Philadelphia, PA, USA, 2006. ACM, New York, NY.
50. J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. GraphScope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*, pages 687–696, San Jose, CA, USA, 2007. ACM, New York, NY.
51. S. R. Sundaresan, I. R. Fischhoff, J. Dushoff, and D. I. Rubenstein. Network metrics reveal differences in social organization between two fission-fusion species, Grevy's zebra and onager. *Oecologia*, 151(1):140–149, 2006.
52. S. R. Sundaresan, I. R. Fischhoff, and D. I. Rubenstein. Male harassment influences female movements and associations in Grevy's zebra (*Equus grevyi*). *Behavioral Ecology*, 18(5): 860–865, 2007.
53. C. Tantipathananandh and T. Y. Berger-Wolf. Constant-factor approximation algorithms for identifying dynamic communities. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining*, pages 827–836, Paris, France, 2009. ACM, New York, NY.
54. C. Tantipathananandh, T. Y. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*, pages 717–726, San Jose, CA, USA, 2007. ACM, New York, NY.
55. H. Tong, S. Papadimitriou, J. Sun, P. S. Yu, and C. Faloutsos. Colibri: fast mining of large static and dynamic graphs. In *Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining*, pages 686–694, Las Vegas, NV, USA, 2008. ACM, New York, NY.
56. M. Toyoda and M. Kitsuregawa. Extracting evolution of web communities from a series of web archives. In *Proceedings of the 14th ACM Conference on Hypertext and Hypermedia*, pages 28–37, Nottingham, U.K., 2003. ACM, New York, NY.
57. V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Germany, 2001.
58. S. Wasserman and K. Faust. *Social Network Analysis.* Cambridge University Press, Cambridge, MA, 1994.
59. T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin. A Bayesian approach toward finding communities and their evolutions in dynamic social networks. In *Proceedings of the 9th SIAM International Conference on Data Mining*, pages 990–1001, Philadelphia, PA, USA, 2009. SIAM.
60. W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

Chapter 13
Structure and Evolution of Online Social Networks

Ravi Kumar, Jasmine Novak, and Andrew Tomkins

Abstract In this work, we consider the evolution of structure within large online social networks. We present a series of measurements of two large real networks, one from the friend relation within the Flickr photo sharing application and the other from Yahoo!s 360 social network. These networks together comprise in excess of 5 million people and 10 million friendship links, and they are annotated with metadata capturing the time of every event in the life of the network. We show that these networks may be segmented into three regions: *singletons*, who do not participate in the network, *isolated communities*, which overwhelmingly display star structure, and a *giant component* anchored by a well-connected core region that persists even in the absence of stars. We give a detailed characterization of the structure and evolution of these regions. We also present a simple model of network growth that captures these aspects of component structure. The model follows our experimental results, characterizing users as either passive members of the network, inviters who encourage offline friends and acquaintances to migrate online, and linkers who fully participate in the social evolution of the network. We show that this simple model with only two numerical parameters is able to produce synthetic networks that accurately reflect the structure of both our real-world networks.

13.1 Introduction

In this work, we study the evolution of large online social networks. While there have been studies of large social networks with fine-grained evolutionary data subsequent to the first appearance of this work, to the best of our knowledge, this is the first detailed evaluation of the growth processes that control online social networks in the large.

R. Kumar (✉)
Yahoo! Research, 701 First Ave, Sunnyvale, CA 94089, USA
e-mail: ravikumar@yahoo-inc.com

Most of this work appeared in the Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining, pp. 611–617, 2006.

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_13,

Online social networks have grown from small-scale curiosities to a global phenomenon that is responsible for a significant fraction of overall Internet pageviews and user engagement. Applications such as Flickr (flickr.com), Myspace (myspace.com), Facebook (facebook.com), and Twitter (twitter.com) have changed the medium through which people interact and have initiated a spirited debate about whether the affordances of these online networks will also change the mechanisms by which people interact. In parallel, online websites in many domains have introduced social capabilities to derive competitive advantage from the massive user populations that frequent them. Broad shopping sites like Amazon (amazon.com), niche marketers like B&H cameras (bhphotovideo.com), auction sites like Ebay (ebay.com) in the United States, Taobao (taobao.cn) in China, and Yahoo! Japan auctions (auctions.yahoo.co.jp) in Japan, and many other domains all benefit from the power of online networks. Likewise, a cornucopia of startup companies have arisen to explore variations on the popular social networking themes, some of which are based on the premise that one's position in various social networks is a valuable asset to be nurtured and grown to promote success in relationships and careers as well as online interactions.

As social networks have grown in scale and visibility, academic interest has kept pace. Offline networks have been the subject of intense academic scrutiny for many decades, but performing such studies at scale is difficult and expensive. The presence of large and easily accessible online social networks allows these detailed offline studies to be augmented with online studies and analyses at massive scale but correspondingly lower resolving power. Initially, many studies of online networks focused on static snapshots of large data sets. In this work, we have access to the entire lifetime of two large social networks, with timestamps mapping each event to the instant of time at which it occurred, and hence we are able to study their dynamic properties at fine detail. We study the social network of Flickr and Yahoo! 360; see Section 13.3.1 for more details about the data sets themselves.

13.1.1 Summary of Findings

We now provide a brief summary of our findings. We begin with a study of the overall properties of the network. We show that the density of the network, which measures the amount of interconnection per person, follows the same unexpected pattern in both networks: rapid growth, decline, and then slow but steady growth. We postulate based on the timing of the events that the pattern is due to the activities of early adopters who create significant linkages in their exploration of the system, followed by a period of rapid growth in which new members join more quickly than friendships can be established, settling finally into a period of ongoing organic growth in which both membership and linkage increase.

Next, we classify members of a social network into one of three groups, the singletons, the giant component, and the middle region, as follows:

Singletons. The singletons are degree-zero nodes who have joined the service but have never made a connection with another user in the social network. They may be viewed as loners who do not participate actively in the network.

Giant component. The giant component represents the large group of people who are connected to one another through paths in the social network. These people find themselves connected directly or indirectly to a large fraction of the entire network, typically containing most of the highly active and gregarious individuals.

Middle region. The middle region is the remainder. It consists of various isolated communities, small groups who interact with one another but not with the network at large. We will show that this group may represent a significant fraction of the total population.

We begin with a detailed study of the middle region, which represent about 1/3 of the users of Flickr and about 10% of the users of Yahoo! 360. We show first that over significant periods of time, and significant fractions of growth in the network (exceeding 10×), the fraction of users who exist in isolated communities of a particular size remains remarkably stable, even though the particular users change dramatically.

We study the migration patterns of isolated communities, seeking insight into how these communities grow and merge. Our findings are quite surprising. The likelihood that two isolated communities will merge is unexpectedly low. Evolution in the middle region is characterized by two processes: isolated communities grow by a single user at a time and then may eventually be merged into the giant component; these processes capture the majority of activity within the middle region. Furthermore, we present a structural finding showing that almost all the isolated communities are in fact *stars*: a single charismatic individual (in the online sense) linked to a varying number of other users who have very few other connections.

We study the formation of these stars and show that they grow rapidly and then either merge into the giant component or cease growth when the individual holding the community together loses focus on growing the network.

Next, we turn to the structure of the giant component. We show that, in this region, the merging of stars does not represent the defining structural characteristic of the giant component. Instead, merging stars represent a sort of outer layer of the region, around a much more tightly connected core of active members who are the heart of the entire social network. Removal of all stars from the giant component has no significant impact on the connectivity of the remaining nodes.

Over time, the average distance between users in the giant component is seen to fall. This surprising phenomenon has been observed in other settings [23]; we show it here for online social networks.

Given these findings, we draw some high-level behavioral conclusions about the structure and evolution of online social networks. First, there are two distinct ways that people join the network: they may register by actively seeking out the network, or they may be invited by a friend or colleague. The stars in the middle region are

largely characterized by invitations, and the individuals performing the invitations are typically motivated more by migrating and existing offline social network into an online setting, rather than building new connections online. On the other hand, the members of the well-connected core of the giant component are the reverse: they are highly focused on the evolution of the internal network of which they are perhaps the key piece.

13.1.2 Model

Based on these observations, we propose a rudimentary model of network evolution in which we attempt to capture the salient properties of our measurements using as small a parameter space as possible. Our model uses a notion of biased preferential attachment that introduces a disparity between the relative ease of finding potential online connections within the giant component, and the relative difficulty of locating potential connections out in the isolated communities. The model accurately reproduces the quantitatively very different component structure of Flickr and Yahoo! 360.

13.1.3 Organization

The chapter is organized as follows. In Section 13.2, we discuss the related work on theoretical and experimental analysis of large-scale social and other related networks. In Section 13.3, we describe our experiments and observations about the Flickr and Yahoo! 360 social networks. In Section 13.4, we outline the biased preferential attachment model for online social network evolution. In Section 13.5, we discuss our findings and outline thoughts for future work. Finally, Section 13.5 concludes the chapter.

13.2 Related work

13.2.1 Experimental Studies

Large real-world graphs such as the World Wide Web, Internet topology, phone call graphs, social networks, email graphs, biological networks, and linguistic networks have been extensively studied from a structural point of view. Typically, these studies address properties of the graph including its size, density, degree distributions, average distance, small-world phenomenon, clustering coefficient, connected components, and community structures. We briefly outline some of the work in this area. Faloutsos, Faloutsos, and Faloutsos [13] made a crucial observation showing that the degree distribution on the Internet follows a power law. Subsequently, an intense body of work followed in both computer science and physics communities aimed at

studying properties of large-scale real-world graphs. Power law degree distributions were also noted on the graph defined by the World Wide Web [4, 21]. Broder et al. [8] studied the World Wide web from a connectivity point of view and showed that it has a large strongly connected component. Several other studies have also shown that the average diameter of the web is quite small [3, 8]. Online friendship and email graphs have been studied in the context of explaining and analyzing friendships [18] and demonstrating the small-world and navigability properties of these graphs [1, 9, 25]. For surveys of analysis of large graphs, the readers are referred to [2, 10, 11, 16, 27, 30, 31].

Many of these above studies were performed on static graphs whereas most real-world graphs are evolving in nature. In fact, there are very few papers that study the evolution of real-world graphs; this is partly because of the difficulty in obtaining temporal information about every node/edge arrival in an evolving real-world graph. A typical way this problem is addressed is to take snapshots of the graph at various points in time and use these snapshots to make inferences about the evolutionary process. This approach was used to study the linkage pattern of blogs and the emergence of bursty communities in the blogspace [19]. Structural properties of different snapshots of the World Wide Web graph was studied by Fetterly et al. and Cho et al. [14, 29]. Leskovec and Faloutsos [23] considered citation graphs and showed that these exhibit densification and shrinking diameters over time. Recently, Leskovec et al. [22] studied social network evolution at a microscopic level by using the maximum likelihood method to analyze the individual node and edge arrival events.

13.2.2 Mathematical Models

A parallel body of work is concerned with developing tractable mathematical models for massive graphs. Because of their evolutionary nature and their power law degree distributions, these graphs cannot be modeled by traditional Erdö–Rényi random graphs [6, 12]. However, there have been a few alternate models that are more faithful to observed properties. One is the so-called configuration model, which chooses a graph uniformly at random from all graphs with a prescribed degree distribution [5, 26, 28]; the degree distribution can be set to match practical observations and is usually a power law. Another approach is to use a generative model to describe the evolution of graphs. A typical example is the copying or the preferential attachment model [4, 20]: nodes arrive one by one and link themselves to a pre-existing node with probability proportional to the degree of the latter. This "rich get richer" principle can be analytically shown to induce power law degree distributions. Kleinberg [15, 17] proposed a model to explain the small-world phenomenon and navigability in social networks; see also [32]. Leskovec and Faloutsos [23] proposed a forest-fire graph model to explain the decreasing diameter phenomenon observed in citation graphs. For a survey of mathematical analysis of some of these models, the readers are referred to [7, 16].

13.3 Measurements

In this section, we detail our study on two online social networks at Yahoo! Each social network is presented as a *directed time graph* $G = (V, E)$, i.e., every node $v \in V$ and directed edge $e_t = \langle u, v \rangle \in E$ in the graph G has an associated time stamp v_t and $\langle u, v \rangle_t$ indicating the exact moment t when the particular node v or the edge $e = \langle u, v \rangle$ became part of the graph [19]. In particular, for any time t, there is a natural graph G_t that comprises all the nodes and edges that have arrived up until time t; here we assume that the end points of an edge always arrive during or before the edge itself. We use timegraph to refer to properties that are specific to the evolution and use graph to refer to the graph G_{Jan2006} as the *final graph*. We note that our study of timegraphs is of much finer granularity than almost all of previous such studies in that we know the *exact* moment of each node/edge arrival.

13.3.1 Data Sets

The data set consists of two online social networks at Yahoo!—Flickr and Yahoo! 360. Each of these social networks is presented as a timegraph. For privacy reasons, all the data used in this work were provided to us after appropriate anonymization. For confidential reasons, we do not specify the exact number of nodes or edges in these timegraphs but only provide a ball-park estimate—this will not in any way affect the presentation of our results or the inferences that can be drawn.

Flickr (flickr.com) is an active and popular online photo sharing and social networking community. Flickr users can upload and tag photos and share them with their friends or publicly. Each user in Flickr can invite a new friend to Flickr or can add a pre-existing Flickr user as a friend. In January 2006, the Flickr timegraph consisted of around 1 million nodes and around 8 million directed edges. The data set we used had the following anonymized information about each Flickr user: the time when the user became a Flickr member and the list of friends he/she has on Flickr, and for each friend, the time when the user befriended the person. Even though we had the entire Flickr timegraph available, for our experiments, we focused only on the evolution of the timegraph since the Flickr website was publicly launched (February 2004); this amounted to about 100 weeks worth of data. We made this decision in order to avoid the initial phase before the public launch when Flickr usage was mostly limited to internal users and the user/friendship addition processes were too skewed to lead to meaningful conclusions.

Yahoo! 360 (360.yahoo.com) is a social networking website that is part of the Yahoo! user network. Users of Yahoo! 360 can add contacts and invite other users to the 360 network. Yahoo! 360 is primarily used to share a blog or photo albums among the friends of a user. In January 2006, the Yahoo! 360 timegraph consisted of around 5 million nodes and around 7 million directed edges. As in Flickr, we used an anonymized timegraph and as before chose to discard the initial segment of

the timegraph in order to filter out pre-launch noise/bias. This resulted in about 40 weeks worth of data.

13.3.2 Basic Timegraph Properties

In this section, we consider three basic properties of these timegraphs. The first property we consider is the *reciprocity* of a directed graph, that is, the fraction of directed edges $\langle u, v \rangle$ such that $\langle v, u \rangle$ also exists in the graph. The goal is to understand the following:

Are friendships reciprocal in online social networks?

The reciprocity of the Flickr final graph is around 70.2% and that of the Yahoo! 360 final graph is around 84%. Thus, friendship edges are highly mutual. In fact, a finer analysis shows that not only are many friendship edges reciprocal but in fact many reciprocal edges are formed almost simultaneously. Figure 13.1 shows for reciprocal edges $\langle u, v \rangle_t$ and $\langle v, u \rangle_{t'}$ in the Flickr final graph, the distribution of $|t - t'|$, i.e., the delay (in days) of the reciprocity. We see that an overwhelming fraction of reciprocal edges arrive within a day of each other. A similar phenomenon is also seen in the Yahoo! 360 final graph. From these observations, we conclude that for the purposes of analysis and for simplicity of exposition, we can pretend that the graph is undirected. So, for the remainder of the chapter, we deal only with undirected graphs and treat the Flickr and Yahoo! 360 graphs to be undirected by removing all uni-directional edges.

Next, we look at the *density* of these graphs, that is, the ratio of undirected edges to nodes, of the timegraphs. In a recent work, Leskovec and Faloutsos [23] observed

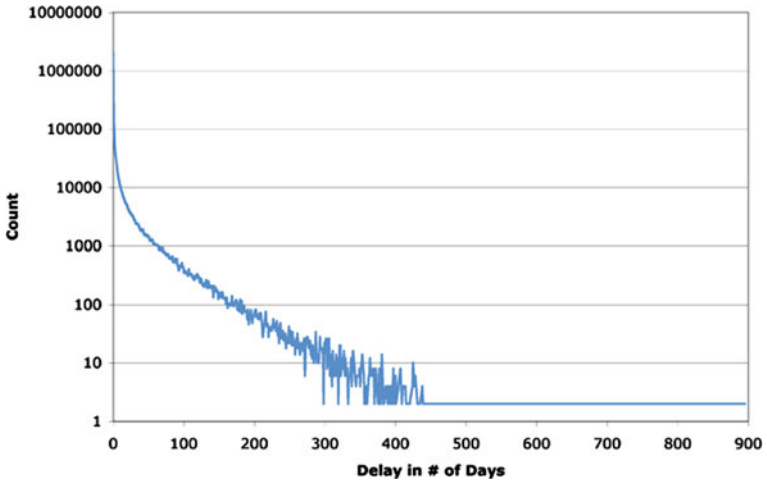


Fig. 13.1 Delay (in days) of reciprocity in Flickr final graph

that certain citation graphs became denser over time. We wish to ask a similar question for online social networks:

How does the density of online social networks behave over time?

It turns out that the density of social networks as a function of time is non-monotone. Figure 13.2 shows the density of the Flickr and Yahoo! 360 timegraphs. In both the plots there are three clearly marked stages: an initial upward trend leading to a peak, followed by a dip, and the final gradual steady increase. We believe that this is due to the following social phenomenon. Right after the launch, there is an initial euphoria among a few enthusiasts who join the network and frantically invite many of their friends to join; this gives rise to the *first stage* that culminates in a peak. The *second stage* corresponds to a natural dying-out of this euphoria and this leads to the dip. The *third stage* corresponds to true organic growth of the network (when more and more people know about the network). This growth takes over the node/edge creation activities, slowly overwhelms the dip, and eventually leads to a steady increase in density. To the best of our knowledge, this phenomenon has not been observed before in real social networks (again, perhaps due to the lack of suitable data).

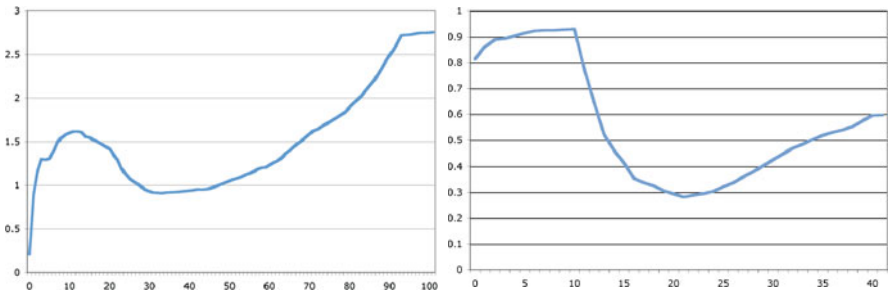


Fig. 13.2 Density of Flickr and Yahoo! 360 timegraphs, by week

For completeness, we also look at the degree distribution of these graphs. Figure 13.3 shows the degree distribution of the Flickr final graph in log–log scale. As expected, it is a power law. The Yahoo! 360 final graph exhibits an almost identical degree distribution. It is interesting to note the non-monotone shape of this plot for the first three values of the degree (i.e., degree = 0, 1, 2). This peak occurs because of the "invite" option that is often used in adding new people to these networks. Typically, many users join via invitation and arrive with a single edge already in place. Degree-zero nodes have explicitly joined the network without an invitation and are a smaller fraction of the total user base. We will return to this issue in Section 13.4.

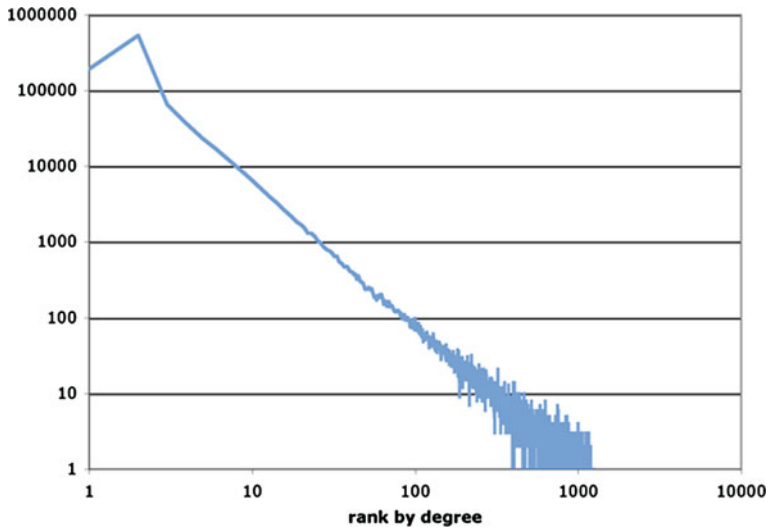


Fig. 13.3 Degree distribution in Flickr final graph. The x-axis is the ranked degree and the y-axis is the number of nodes at this rank

13.3.3 Component Properties

In this section, we study the component structure of the graph in detail. Our goal is to understand the connectivity structure of the graph as it evolves over time. In particular, we ask

What is the dynamics of component formation and evolution in social networks?

We apply a simple connected components algorithm on the timegraph by considering the instance at every week. The results for the Flickr and Yahoo! 360 timegraphs are in Figure 13.4. This plot shows the fraction of nodes in components of various sizes. The intervals representing various horizontal bands were chosen so that the top band represents the largest connected component, which we will call the *giant component*, while the bottom band represents the total number of *singleton* nodes in

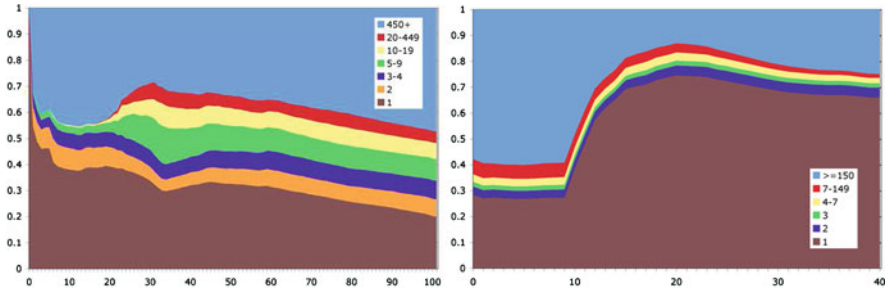


Fig. 13.4 Fraction of nodes in components of various sizes within Flickr and Yahoo! 360 timegraph, by week

the graph, with no links in the social network at all. The rest of the bands constitute the *middle region*, consisting of nodes that exist in small isolated neighborhoods. While there are quantitative differences between the plots for Flickr and Yahoo! 360, both the plots share two particularly interesting properties.

1. The fraction of singletons, the fraction of nodes in the giant component, and fraction of nodes in the middle region remain almost constant once a steady state has been reached, despite significant growth of the social network during the period of steady component structure. For example, the Flickr social network grew by a factor of over 13× from the period $x = 40$ to $x = 100$ in Fig. 13.4, with very little visible change in the fraction of users who occupied components of a certain size. This steady state corresponds to the third stage observed in Fig. 13.2.
2. In the middle region, each band of the diagram appears fairly constant. In fact, as Fig. 13.5 shows, the component size distribution for both data sets follows a power law with exponent −2.74 for the Flickr graph and −3.60 for Yahoo! 360.

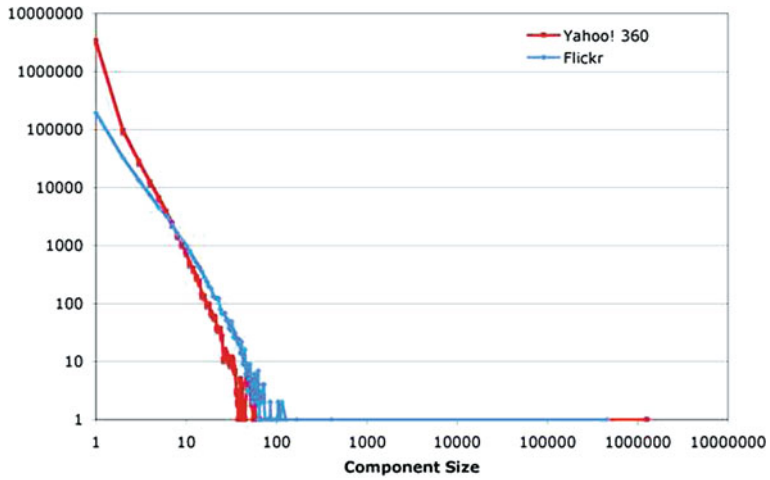


Fig. 13.5 Component size distribution for Flickr and Yahoo! 360 final graph

13.3.4 Structure of the Middle Region

We now proceed to investigate the formation and structure of the middle region. Our first question was motivated by the evolutionary aspect of the timegraph:

How do components merge with each another as nodes and edges arrive in social networks?

In particular, it was our assumption when we began this experiment that the non-giant components would grow organically, with a size 3 component linking to a size

4 component to form a new component of size 7, and so forth. Table 13.1 shows how component merges happen in both Flickr and Yahoo! 360 timegraphs. The (i, j)th entry of this symmetric table gives the number of times during the evolution of the timegraph that a component of size i merges with a component of size j.

Table 13.1 Sizes of components in Flickr and Yahoo! 360 timegraphs when merging, in thousands of nodes

	1	2	3–4	5–9	10–19	20–449	450+
1	205.1						
2	55.9	0.8					
3–4	64.2	0.5	0.3				
5–9	70.8	0.4	0.3	0.2			
10–19	43.9	0.2	0.1	0.1	0.09		
20–449	2.6	0.1	0.01	0.07	0.04	0.03	
450+	315.3	11.5	7.1	5.0	2.4	1.0	0

	1	2	3	4	5–7	8–149	150+
1	584.3						
2	126.1	5.9					
3	69.2	2.6	1.2				
4	43.6	1.5	0.6	0.4			
5–7	66.9	2.3	1.0	0.6	0.9		
8–149	72.6	2.3	1.1	0.6	0.9	1.1	
150+	767.3	54.9	22.4	12.2	15.7	13.0	0.1

Strikingly, almost all the mass in this table is in the bottom row and the left column, implying that the component merges are of primarily two types:

- singletons merging with the current non-giant components and the giant component and
- non-giant components, including singletons, merging with the giant component.

That is, it is surprisingly rare during the evolution of the timegraph that two non-giant components merge to produce another non-giant component.

Our next goal is to understand the consequences of this observed phenomenon and its impact on the structure of the middle region. Indeed, if most of the component merges are characterized by the above two types, it is natural to speculate that this is caused by some special node in the non-giant component that serves to "attract" the incoming singleton. Notice that if this were to happen, it would lead to many middle region *stars*, that is, components with a center of high degree and many low-degree nodes connected to the center. We ask,

> *Do the components in the middle region have any special structure, and in particular, are they stars?*

First, to be able to observe this phenomenon, we need a reasonably robust definition of a star. We define a star to be connected component with the following two properties: it has one or two nodes (centers) that have an edge to most of the other nodes in the component and it contains a relatively large number of nodes that have an edge solely to one of these centers. More formally, let U be the nodes in a connected component that is not the giant component. Trivially, U is a star if $|U| = 2$. Otherwise, let $C \subseteq U$ be the set of nodes with degree more than $|U|/2$ and let $T \subseteq U$ be the set of nodes with degree equal to one. For a parameter $k \in (0, 1)$, we define U to be a *star* if $|C| \in \{1, 2\}$ and $|T|/|U \setminus C| > k$; we call C the *centers* of the star and $|T|$ the *twinkles*. In our experiments, we set $k = 0.6$ in the above definition.

Based on this definition of a star, we analyze the final graphs of both Flickr and Yahoo! 360. In the Flickr final graph, 92.8% of the middle region was composed of stars; in total there were 69,532 centers and 222,564 twinkles. In the Yahoo! 360 final graph, 88.7% of the middle region was composed of stars; there were 147,071 centers and 264,971 twinkles. Thus, there is an overwhelming number of stars in the middle region, validating our hypothesis that each component in the middle region has a center and the singleton node joins the center to become a twinkle. We will make heavy use of this characterization in order to develop a generative model that produces an appropriate middle region.

In fact, our hypothesis is further strengthened when we examine this process more closely. Call a star *non-trivial* if it has more than two nodes and let u be the center of a non-trivial star. Figure 13.6 shows the distribution of the time lag between first twinkle u and the last twinkle u' to join the star, i.e., the distribution of $t' - t$ (in weeks) where $\langle u, v\rangle_t$ is the edge that adds the first twinkle and $\langle u', v\rangle_{t'}$ is the edge that adds the last twinkle. As we see, the distribution is sharply decreasing, suggesting that stars are formed rather quickly. We next analyze the *age* of stars, which is the time since the last edge arrival in the star. Figure 13.7 shows the age of stars in the Flickr final graph. Again, a large fraction of stars are more than 10 weeks old. This suggests that the middle section consists of stars that are formed quickly but have not been absorbed into the giant component yet.

Similar results were also observed for Yahoo! 360 final graph. For sake of brevity, we do not present these results.

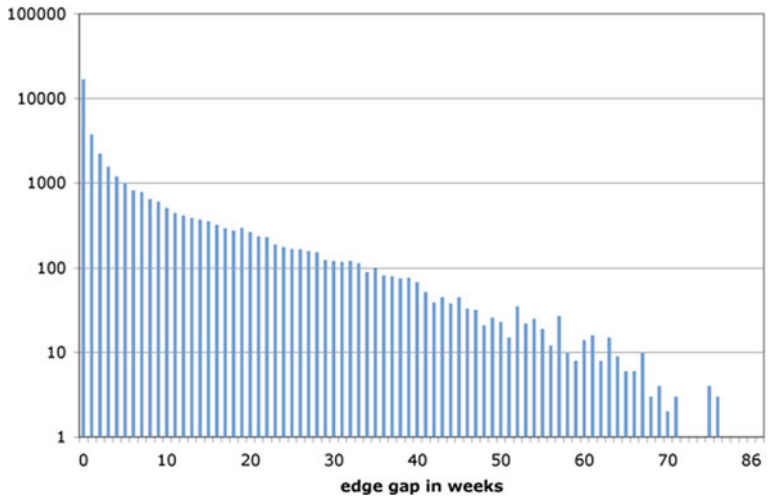


Fig. 13.6 Distribution of time lag (in weeks) between the first and last twinkle addition to non-trivial stars in the Flickr final graph

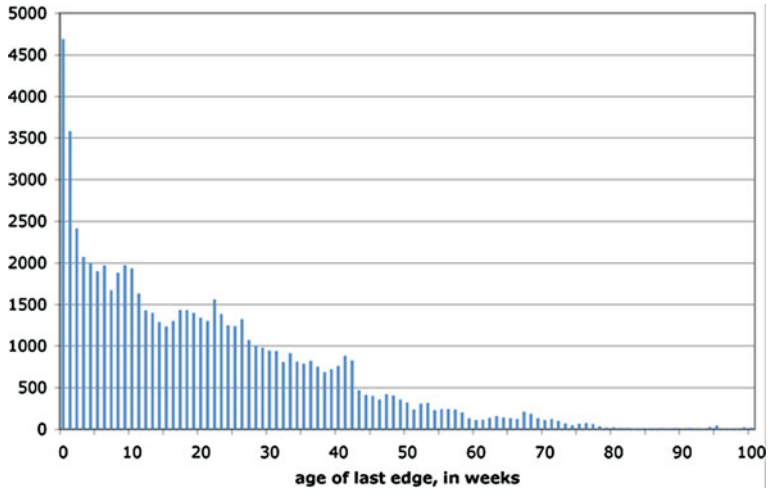


Fig. 13.7 Age of non-trivial stars in the Flickr final graph

13.3.5 Structure of the Giant Component

In this section, we analyze the structure of the giant component. The most natural question to ask is,

How does the diameter of the social network behave as a function of time?

We study the diameter of the giant component. Formally, the diameter is the maximum over all pairs in the giant component of the shortest path connecting the pair. This measure is not robust in general, as a single long path in the component could result in an enormous diameter. Thus, we turn instead to the *average diameter*, which is defined as the length of the shortest path between a random pair of nodes. For comparison, we also consider the *effective diameter*, which is defined as the 90th percentile of the shortest path lengths between all pairs of nodes; this quantity was used in [23]. We estimate both these quantities by sampling sufficiently many pairs of nodes in the giant component uniformly at random.

For the giant component in the Flickr final graph, we compute the average diameter to be 6.01 and the effective diameter to be 7.61. For the giant component in the Yahoo! 360 final graph, the corresponding values are 8.26 and 10.47, respectively. Notice that these are slightly higher values than the one suggested by the "six-degrees of separation" folklore. Figure 13.8 shows diameter as a function of time in the Flickr and Yahoo! 360 timegraphs. The shape of this curve has high correlation with that of density over time, which exhibited three distinct stages in the evolution of the timegraph. We note that the three stages in Fig. 13.8 exactly correspond to the three stages in Fig. 13.2. In the first stage, the diameter is almost flat. In the next stage, where the edge density drops, the diameter grows till it reaches a peak. In the third stage, when the edge density starts increasing, the diameter starts decreasing.

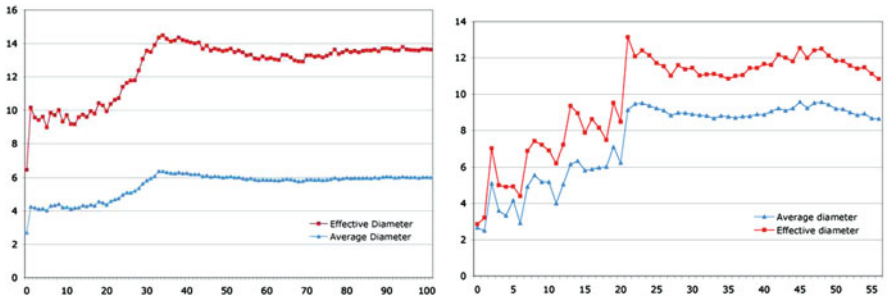


Fig. 13.8 Average and effective diameter of the giant component of Flickr and Yahoo! 360 timegraphs, by week

A similar phenomenon of shrinking diameter was recently observed by Leskovec and Faloutsos [23] in citation graphs. Our study shows that diameter shrinking happens in social networks as well. Again, to the best of our knowledge, this is the first instance of such an observation for online social networks. Well-known models of network growth based on preferential attachment [4, 20] do not have this property (see [7] for details).

We then investigate the structure to see if we can explain the diameter values that were observed. In particular, we ask,

Does the giant component have a reasonably small core of nodes with high connectivity?

By computing the degree distribution of the nodes in the giant component, we observe that in the Flickr final graph, about 59.7% of the nodes in the giant component have degree one. The corresponding number for the Yahoo! 360 final graph was 50.4%. These degree-one nodes therefore contribute to the increase in diameter values. Suppose we discard these degree-one nodes in the giant component and analyze the remaining *one-pruned subgraph*. For the one-pruned subgraph of the Flickr final graph, the average diameter is 4.45 and the effective diameter is 5.58. For the one-pruned subgraph of the Yahoo! 360 final graph, the corresponding numbers are 6.52 and 7.95, respectively. This suggests that there is a subgraph inside the giant component of extremely high connectivity.

To explore this question in more detail, we study the *k-cores* of the giant component. The k-core of a graph is produced by iteratively removing all nodes with degree less than or equal to k until no more such nodes remain. Notice that, after removing a node, a degree-two neighbor of that node would now have degree one and would also need to be removed, hence the iterative nature of the definition.

For each of our networks, we study the number of nodes and edges and also the average degree of the k-core for a range of values of k. Figure 13.9 shows the size of the resulting k-cores in nodes and edges for k ranging from 1 to 10. Node counts are on the left axis and edge counts on the right. We begin by considering the data for the Flickr graph, in Figure 13.9. The one-core retains around 1/3 of the nodes of the original giant component, and even the five-core, which represents an aggressive pruning, retains about 1/3 of the nodes of the one-core. The edges drop much more

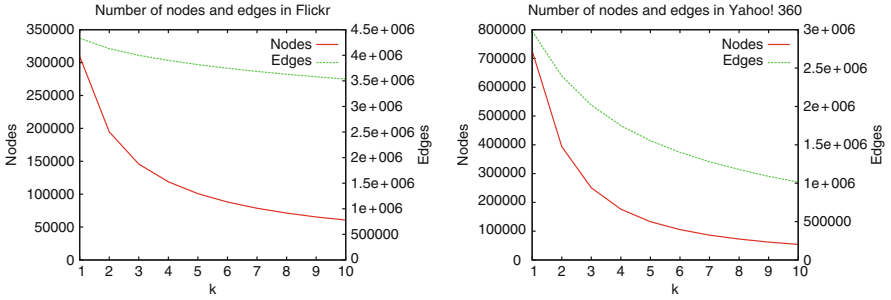


Fig. 13.9 Nodes and edges in k-core for Flickr and Yahoo! 360 timegraphs as a function of k

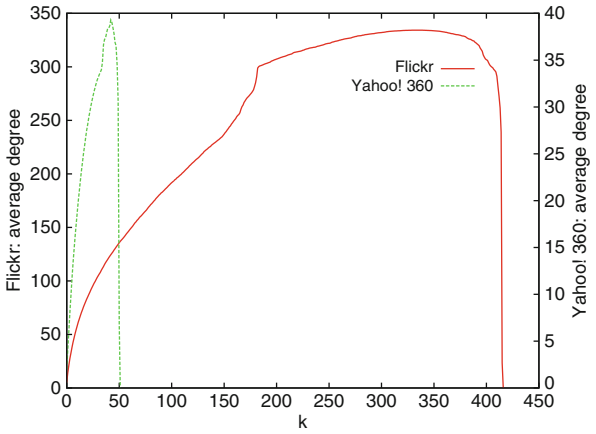


Fig. 13.10 Average degree of k-core for Flickr and Yahoo! 360 timegraphs as a function of k

slowly: even after computing the 10-core, about 80% of the edges in the original one-core still remain. Thus, the internal connectivity structure is extremely strong.

The Yahoo! 360 graph shows a similar pattern, but with less strong connectivity than Flickr. The five-core shows about 18% of the nodes in the one-core remaining, and the 10-core retains 30% of the edges of the one-core.

Figure 13.10 shows the average degree for a much broader range of values of k. The Flickr core retains over 1000 nodes up to $k = 400$, showing that there is a strong community of extremely densely connected nodes at the heart of the giant component. The Yahoo! 360 graph fragments at $k = 50$, and a core of 1000 nodes persists up to $k = 45$.

Stars are the dominant explanation of the structure outside the giant component. Given the presence of this small core of well-connected nodes, one might naturally ask the following question:

Are stars merging into the giant component also responsible for the highly connected core of the giant component?

We identify all stars throughout the life of the time graph and track them as they merge into the giant component. Based on this tracking, we remove all star centers, and both the original twinkles belonging to that star, and all new degree-one nodes connected to that star, and ask whether any fragmentation results. In fact, the giant component remains extremely well connected. (This phenomenon was also independently and subsequently observed by Leskovec et al. [24].)

Thus, we conclude that the stars represent the primary form of structure outside the giant component, but represent only a thin layer of structure at the outside of the giant component. The true characteristic of the giant component is the well-connected core at the center. Later we will discuss some possible implications of this observation.

13.4 Model

In this section, we present a model of the evolution of online social networks. Our goal in developing this model is to explain the key aspects of network growth in as simple a manner as possible, obviating the need for more complex behavioral explanations.

13.4.1 Desiderata

The properties we will seek to reproduce are the following.

Component structure. The model should produce an evolving component structure similar to that of Fig. 13.4. The fraction of users who are singletons, those in the middle region, and those in the giant component should reflect the underlying data. The non-giant component of each size should capture a fraction of the users that matches the empirical observations and should analytically match the observed power law.

Star structure. The non-giant components should be predominantly star-like. Their growth rates should match the growth of the actual data.

Giant component structure. The nodes making up the giant component should display a densely connected core and a large set of singleton hangers-on, and the relationship between these regions should explain the average distance of the giant component.

13.4.2 Description of the Model

Our model is generative and informally proceeds as follows. There are three types of users: passive, linkers, and inviters. *Passive users* join the network out of curiosity or at the insistence of a friend, but never engage in any significant activity. *Inviters*

are interested in migrating an offline community into an online social network and actively recruit their friends to participate. *Linkers* are full participants in the growth of the online social network and actively connect themselves to other members.

At each timestep, a node arrives and is determined at birth to be passive, linker, or inviter according to a coin toss. During the same timestep, ε edges arrive and the following happens for each edge. The source of the edge is chosen at random from the existing inviters and linkers in the network using preferential attachment; that is, the probability that a particular node is chosen is proportional to its degree plus a constant. If the source is an inviter, then it invites a non-member to join the network, and so the destination is a new node. If the source is a linker, then the destination is chosen from among the existing linkers and inviters, again using preferential attachment. The parameters controlling the model are shown below.

	Description of the parameter
p	User type distribution (passive, inviter, linker)
γ	Preference for giant component over the middle region
ε	Edges per timestep

More formally, the model proceeds as follows. We incrementally build a time-graph $G = (V, E)$. At any point in time, let the set of passives, inviters, and linkers be denoted by P, I, and L respectively, such that $V = P \cup I \cup L$. Let $d(u)$ denote the degree of node u.

At each timestep, a new node arrives and is assigned to P, I, or L according to the probabilities in p. Let $\beta > 0$ be a parameter. We will define probability distribution D^β over V representing the probability of selecting a node u via a *biased preferential attachment*, as follows:

$$D^\beta(u) \propto \begin{cases} \beta \cdot (d(u) + 1) & u \in L \\ d(u) + 1 & u \in I \\ 0 & \text{otherwise.} \end{cases}$$

Then ε undirected edges arrive, as follows. For each edge (u, v), u is chosen from D^1, where the bias parameter is set to 1. If u is an inviter, then v is a new node, assigned to P. If u is a linker then v is chosen from D^γ. Notice that the initiator of a link is chosen from all non-passive nodes based only on degree. However, once a linker decides to generate a node internal to the existing network, the destination of that node is biased toward other linkers by γ. This reflects the fact that the middle region is more difficult to discover when navigating a social network.

13.4.3 Simulations

We now evaluate the model with respect to the three families of conditions we hope it will fulfill. We choose suitable parameters for our model and simulate the model.

Table 13.2 Parameter choices for Flickr and Yahoo! 360

	p (passive, inviter, linker)	γ	ε
Flickr	(0.25, 0.35, 0.40)	15	6
Yahoo! 360	(0.68, 0.22, 0.10)	2	1

We then examine the properties of the graph created by our model and see how closely it matches that of Flickr and Yahoo! 360 timegraphs. Table 13.2 shows the appropriate parameter choices.

We refer to the graphs generated by simulation as Flickr.model and 360.model. We start with the component structure of these simulations and compare them against the actual data. The actual fraction of nodes in each of the three main regions (0.20, 0.33, 0.47 for Flickr and 0.66, 0.09, 0.25 for Yahoo! 360) is exactly matched in the graph obtained by simulation.

We now refine the middle region further and compare the simulated versus the actual data. Table 13.3 shows the results. From our simulation, we see that in terms of components and the structure of the middle region, our model can accurately capture the properties of Flickr and Yahoo! 360 graphs, when the parameters are well chosen.

Table 13.3 Middle region in actual and simulated data

Flickr	1	2	3–4	5–9	10–19	20–449	≥ 450	Yahoo! 360	1	2	3	4–6	7–149	≥ 150
Actual	0.2	0.07	0.07	0.08	0.06	0.05	0.47	Actual	0.66	0.038	0.016	0.02	0.016	0.25
Model	0.2	0.06	0.08	0.08	0.06	0.03	0.47	Model	0.66	0.04	0.02	0.02	0.01	0.25

13.5 Discussions and Future Work

There are several key takeaway points from our experiments. The first is that online social networks often contain more than half their mass outside the giant component, and the structure outside the giant component is largely characterized by stars. The creation of stars is largely a result of the dynamics of invitation, in which many people are invited to the social network, but only a small fraction choose to engage more deeply than simply responding to an invitation from a friend.

The second key takeaway is that online social networks appear to travel through distinct stages of growth, characterized by specific behavior in terms of density, diameter, and regularity of component structure. We have observed these changes by studying the time graphs of two very different social networks, but we do not yet have a more detailed characterization of the root cause for this progression. It would be attractive to develop a more detailed theory of the adolescence of a social network.

Third, Fig. 13.4 shows a surprising macroscopic component structure in which the total mass of individuals is well spread across a broad range of sizes of isolated communities (or from a graph theoretic perspective, smaller components). We feel

that a deeper understanding of the behavior of "middle band" activity versus "core" activity may reveal that the dichotomy is a meaningful reflection of two active by very different types of participants.

Finally, we have presented a simple model that is surprisingly accurate in its ability to capture component growth. It will be interesting to do a more detailed analysis of the model to show that it also predicts diameter of the giant component, in addition to structure of the middle region. Similarly, the model itself is optimized to be the simplest possible approach to reproducing particular aspects of social network structure rather than a detailed model built from the data in order to provide predictive power. Nonetheless, it is interesting to ask whether the best fitting model parameters may be taken as descriptive of the social network in any sense. For example, in the model, Yahoo! 360 displays a smaller relative fraction of active members, compared to the Flickr community, but at the same time offers fewer barriers to discovering isolated sub-communities and incorporating them into the giant component. Is this representative of the underlying reality?

13.6 Conclusions

We studied the structure and evolution of two popular online social networks, namely Flickr and Yahoo! 360. Our study analyzes these graphs from an evolutionary point of view, by keeping track the precise moments when each node and edge arrives in the graph. We show that these quantitatively different graphs share many qualitative properties in common. In particular, we analyzed the structure and evolution of different-sized components and showed the prevalence of "stars," an intriguing feature of online social networks. Based on these empirical observations, we postulated a very simple evolving graph model for social networks and showed by simulation that this model faithfully reflects the observed characteristics. Since our model is fairly simple, we believe it is amenable to mathematical analyses.

Our work raises a number of questions about the behavioral characteristics of the users who contribute to these various different network regions.

Acknowledgments We are grateful to the Flickr and Yahoo! 360 teams at Yahoo! for their support in data gathering, data analysis, and direction. In particular, we would like to thank Stewart Butterfield, Catarina Fake, Serguei Mourachov, and Neal Sample.

References

1. L. A. Adamic and E. Adar. How to search a social network. *Social Networks*, 27(3):187–203, 2005.
2. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74: 47, 2002.
3. R. Albert, H. Jeong, and A.-L. Barabasi. Diameter of the world wide web. *Nature*, 401: 130–131, 1999.

4. A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
5. B. Bollobás. A probabilistic proof of an asymptotic formula for the number of labeled regular graphs. *European Journal of Combinatorics*, 1:311–316, 1980.
6. B. Bollobas. *Random Graphs*. Cambridge University Press, Cambridge, UK, 2001.
7. B. Bollobas and O. Riordan. Mathematical results on scale-free random graphs, In S. Bornholdt and H. G. Schuster, editors, *Handbook of Graphs and Networks*, pages 1–34. Wiley-VCH, Weinheim, Germany, 2002.
8. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the web. *WWW9/Computer Networks*, 33(1–6):309–320, 2000.
9. P. S. Dodds, R. Muhamad, and D. J. Watts. An experimental study of search in global social networks. *Science*, 301:827–829, 2003.
10. S. Dorogovtsev and J. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford University Press, Oxford, England, 2000.
11. S. Dorogovtsev and J. Mendes. Evolution of networks. *Advances in Physics*, 51:1079–1187, 2002.
12. P. Erdös and A. Rényi. On random graphs I. *Publications Mathematics Debrecen*, 6:290–297, 1959.
13. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of ACM SIGCOMM Conference*, pages 251–262, Cambridge, MA, Aug 1999.
14. D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. *Software Practice and Experience*, 34(2):213–237, 2004.
15. J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, Portland, OR, May 2000.
16. J. Kleinberg. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians*, pages 1019–1044, Madrid, Spain, Aug 2006.
17. J. M. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.
18. R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. Structure and evolution of blogspace. *Communications of the ACM*, 47(12):35–39, 2004.
19. R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. *World Wide Web Journal*, 8(2):159–178, 2005.
20. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of 41st Annual Symposium on Foundations of Computer Science*, pages 57–65, Redondo Beach, CA, Nov 2000.
21. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. *WWW8/Computer Networks*, pages 1481–1493, Toronto, Canada, May 1999.
22. J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining*, pages 462–470, Las Vegas, NV, Aug 2008.
23. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters, and possible explanations. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining*, pages 177–187, Chicago, IL, Aug 2005.
24. J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web*, pages 695–704, Beijing, China, Apr 2008.
25. D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins. Geographic routing in social networks. *Proceedings of the National Academy of Sciences*, 102(33):11623–11628, 2005.
26. M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, 6:161–180, 1995.

27. M. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.
28. M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64(2):026118(17 pages), 2001.
29. A. Ntoulas, J. Cho, and C. Olston. What's new on the web? the evolution of the web from a search engine perspective. In *Proceedings of the 13th International Conference on World Wide Web*, pages 1–12, New York, NY, May 2004.
30. S. Strogatz. Exploring complex networks. *Nature*, 410:268–276, 2001.
31. S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, UK, 1994. Revised, reprinted edition, 1997.
32. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

Chapter 14
Toward Identity Anonymization in Social Networks

Kenneth L. Clarkson, Kun Liu, and Evimaria Terzi

Abstract The proliferation of network data in various application domains has raised privacy concerns for the individuals involved. Recent studies show that simply removing the identities of the nodes before publishing the graph/social network data does not guarantee privacy. The structure of the graph itself, and in its basic form the degree of the nodes, can be revealing the identities of individuals. To address this issue, we study a specific graph-anonymization problem. We call a graph k-degree anonymous if for every node v, there exist at least k-1 other nodes in the graph with the same degree as v. This definition of anonymity prevents the re-identification of individuals by adversaries with a priori knowledge of the degree of certain nodes. We formally define the graph-anonymization problem that, given a graph G, asks for the k-degree anonymous graph that stems from G with the minimum number of graph-modification operations. We devise simple and efficient algorithms for solving this problem. Our algorithms are based on principles related to the realizability of degree sequences. We apply our methods to a large spectrum of synthetic and real data sets and demonstrate their efficiency and practical utility.

14.1 Introduction

Social networks, online communities, peer-to-peer file sharing, and telecommunication systems can be modeled as complex graphs. These graphs are of significant importance in various application domains such as marketing, psychology, epidemiology, and homeland security. The management and analysis of these graphs is a recurring theme with increasing interest in the database, data mining, and theory communities. Past and ongoing research in this direction has revealed interesting properties of the data and presented efficient ways of maintaining,

E. Terzi (✉)
Computer Science Department, Boston University, Boston, MA, USA
e-mail: evimaria@cs.bu.edu

This work was done when the authors Kun Liu and Evimaria Terzi were at IBM Almaden Research Center.

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_14,

querying, and updating them. The proliferation of social networks has inevitably raised issues related to privacy-preserving data analysis as illustrated in recent papers, e.g., [2, 11, 18, 22, 23].

Compared with existing anonymization and perturbation techniques of tabular data (see, e.g., the survey book [1]), working with graphs and networks is much more challenging. Some aspects of graph data that enhance the challenge are the following:

- It is difficult to model the background knowledge and the capability of an attacker. Any topological structures of the graph can be exploited by the attacker to derive private information. Two nodes that are indistinguishable with respect to some structural metrics may be distinguishable by other metrics.
- It is difficult to quantify the information loss. A graph contains rich information but there is no standard way to quantify the information loss incurred by the changes of its nodes and edges.
- It is even difficult to devise graph-modification algorithms that balance the goals of preserving privacy with the utility of the data. Although in tabular data where each tuple can be viewed as an independent sample from some distribution, the nodes and edges in a graph are all related to each other. Therefore, the impact of a single change of an edge or a node can spread across the whole network.

Recall that in a social network, nodes correspond to individuals or other social entities and edges correspond to social relationships between them. The privacy breaches in social network data can be grouped into three categories: (1) *identity disclosure*: the identity of the individual who is associated with the node is revealed; (2) *link disclosure*: sensitive relationships between two individuals are disclosed; and (3) *content disclosure*: the privacy of the data associated with each node is breached, e.g., the email message sent and/or received by the individuals in an email communication graph. A perfect privacy-protection system should consider all of these issues. However, protecting against each of the above breaches may require different techniques. For example, for content disclosure, standard privacy-preserving data mining techniques [1], such as data perturbation and k-anonymization, can help. For link disclosure, the various techniques studied by the link-mining community [9, 23] can be useful.

The techniques presented in this chapter aim toward protection of *identity disclosure* of individuals in a social network. In their recent work, Backstrom et al. [2] point out that the simple technique of anonymizing graphs by removing the identities of the nodes before publishing the actual graph does not always guarantee privacy. It is shown in [2] that there exist adversaries that can infer, in polynomial time, the identity of the nodes using graph-isomorphism algorithms designed for a restricted class of graphs. However, the problem of designing techniques that could protect individuals' privacy has not been addressed in [2].

Motivated by [2], we focus our attention on protecting the identities of individuals from adversaries that have prior knowledge of the degrees of some nodes. As an example, consider a social network graph and an adversary who connects to a node

with a degree x that is unusually high. It would be rather unexpected for there to be another node of degree exactly x. Now assume a "naive" privacy-protection strategy that simply removes the names of the nodes in the network before releasing the graph. Then it is obvious that if degree x is unique in the graph, then the adversary can re-identify his high-degree neighbor by simply asking the query "Find all nodes with degree x." In this chapter we present methods that prevent this. For that we describe a k-anonymity notion for graphs that is similar to the k-anonymity notion introduced for tabular data [19]. In other words, this chapter describes a methodology for answering the following question: *How can a graph be minimally modified to protect the identity of each individual involved?*

The material presented here is an extension of our earlier work presented in [15]. Since the publication of our original paper, several other methods have been developed for identity anonymization on graphs. We give a summary of these methods in the next section.

14.2 Related Work

Since the introduction of the concept of anonymity in databases [19], there has been increasing interest in the database community in studying the complexity of the problem and proposing algorithms for anonymizing data records under different anonymization models [4, 16, 17]. Though much attention has been given to the anonymization of tabular data, the privacy issues of graphs and social networks, and the notion of anonymization of graphs, have only been recently touched.

Backstrom et al. [2] were the first to study attacks against simple privacy-preserving methods for social networks. In their seminal paper ([2]) they show that simply removing the identifiers of the nodes does not always guarantee privacy. Adversaries can infer the identity of the nodes by solving a set of restricted isomorphism problems based on the uniqueness of small random subgraphs embedded in a network.

As we have already discussed in the introduction, this observation gave rise to three types of risks of challenges in privacy-preserving data mining methods: how to prevent identity disclosure, link disclosure, and content disclosure in social networks. To combat these challenges, several authors have recently developed different types of privacy models, adversaries, and graph-modification algorithms. Unfortunately, none of the work is likely to solve all the problems in one shot. Protecting against each kind of privacy breach may require different techniques or a combination of them. Below we give a list of papers dealing with these challenges. This list should be conceived as indicative rather than complete. For a more thorough review of the literature, see [24].

Apart from our work in [15] which we describe here, the problem of identity anonymization in social networks has been studied in [11, 18].

Hay et al. [11] observe that the structural similarity of the nodes in the graph determines the extent to which an individual in the network can be distinguished from others. Based on the notion of k-anonymity [19], Hay et al. [11] proposed a

scheme of anonymity through structural similarity. Vertices that look structurally similar may be indistinguishable to an adversary. A strong form of structural similarity between vertices is automorphism equivalence. The anonymization technique proposed in [11] is a node-clustering approach. It generalizes an input network by grouping vertices into partitions and publishing the number of vertices in each partition along with the densities of edges within and across partitions. Data analysts can still use the anonymized graphs to study macro-properties of the original graph.

Pei and Zhou in [18] consider yet another definition of graph anonymity: a graph is k-anonymous if for every node there exists at least $k-1$ other nodes that share isomorphic neighborhoods; in this case the neighborhood of a node is defined by its immediate neighbors and the connections between them. This definition of anonymity in graphs is different from ours. In a sense it is a more strict one.

Protection of links between individual graph entities has been studied in [13, 22, 23]. Zheleva and Getoor [23] consider the problem of protecting sensitive relationships among the individuals in the anonymized social network. This is closely related to the link-prediction problem that has been widely studied in the link-mining community [9]. In [23] simple edge-deletion and node-merging algorithms are proposed to reduce the risk of sensitive link disclosure.

Sensitive link and relationship protection is also discussed by Ying and Wu [22]. They study how anonymization algorithms that are based on randomly adding and removing edges change certain graph properties. More specifically, they focus on the change caused in the eigenvalues (spectrum) of the network. The authors additionally explore how the randomized network can be exploited by an adversary to gain knowledge about the existence of certain links.

Korolova et al. [13] considered the problem where an attacker wants to derive the link structure of the entire network by collecting the neighborhood information of some compromised users, who are either bribed or whose accounts are broken into by the attacker. Analysis shows that the number of users needed to be compromised in order to cover a constant fraction of the entire network drops exponentially with increase in a lookahead parameter ℓ. Parameter ℓ determines if a registered user can see all of the links and nodes within distance ℓ from him.

Content disclosure is normally an issue when the private data associated with a user on the network is disclosed to others. A very interesting example recently arose from Facebook's "Beacon" service, a "social ads" system where your own expressed brand preferences and Internet browsing habits, and even your very identity, are used to market goods and services to you and your friends. For example, adding the latest season of LOST to your queue on Blockbuster.com might result in Facebook placing an ad for Blockbuster straight on your friends' news feeds. This helps Facebook and its partners (Blockbuster in this example) make money because, as Facebook's CEO Mark Zuckerberg extols, "nothing influences a person more than a recommendation from a trusted friend." This may be fine in some situations, but there may be some things that one is not prepared to share with the entire world. From the users' perspective, they want to ask how to avoid the disclosure of their personal private information while still enjoying the benefit of social advertisement. Companies on the other hand want to assure the users that their privacy is not compromised while

doing social advertisement. Privacy concerns regarding content disclosure exist in other application scenarios such as social recommendation. Protecting against this kind of disclosure is an important research and engineering problem. However, the work in the literature thus far does not take into account how graph structures affect the content disclosure; they rather focus on standard data perturbation and anonymization for tabular data.

14.3 Problem Definition

Let $G(V, E)$ be a simple graph; V is a set of nodes and E the set of edges in G. We use $\mathbf{d}_G$ to denote the *degree sequence* of G. That is, $\mathbf{d}_G$ is a vector of size $n = |V|$ such that $\mathbf{d}_G(i)$ is the degree of the ith node of G. Throughout the chapter, we use $\mathbf{d}(i)$, $\mathbf{d}(v_i)$, and $\mathbf{d}_G(i)$ interchangeably to denote the degree of node $v_i \in V$. When the graph is clear from the context we drop the subscript and use $\mathbf{d}(i)$ instead. Without loss of generality, we also assume that entries in $\mathbf{d}$ are in nonincreasing order, that is, $\mathbf{d}(1) \geq \mathbf{d}(2) \geq \cdots \geq \mathbf{d}(n)$. Additionally, for $i < j$ we use $\mathbf{d}[i, j]$ to denote the subsequence of $\mathbf{d}$ that contains elements $i, i+1, \ldots, j-1, j$.

Before defining the notion of a *k-degree anonymous graph*, we first define the notion of a k-anonymous vector of integers.

Definition 1 A vector of integers $\mathbf{v}$ is k-anonymous if every distinct value in $\mathbf{v}$ appears at least k times.

For example, vector $\mathbf{v} = [5, 5, 3, 3, 2, 2, 2]$ is 2-anonymous.

Definition 2 A graph $G(V, E)$ is k-degree anonymous if its degree sequence $\mathbf{d}_G$ is k-anonymous.

Alternatively, Definition 2 implies that for every node $v \in V$ there exists at least $k-1$ other nodes that have the same degree as v. This property prevents the re-identification of individuals by adversaries with a priori knowledge of the degree of certain nodes. This echoes the observation made by Hay et al. [12].

Figure 14.1 shows two examples of degree anonymous graphs. In the graph on the left, all three nodes have the same degree and thus the graph is 3-degree anonymous. Similarly, the graph on the right is 2-degree anonymous since there are two nodes with degree 1 and four nodes with degree 2.

Degree anonymity has the following monotonicity property.

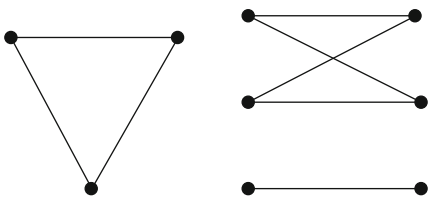


Fig. 14.1 Examples of a 3-degree anonymous graph (*left*) and a 2-degree anonymous graph (*right*)

Proposition 1 *If a graph $G(V, E)$ is k_1-degree anonymous, then it is also k_2-degree anonymous, for every $k_2 \leq k_1$.*

We use the definitions above to define the GRAPH ANONYMIZATION problem. The input to the problem is a simple graph $G(V, E)$ and an integer k. The requirement is to use a set of graph-modification operations on G in order to construct a k-degree anonymous graph $\widehat{G}(\widehat{V}, \widehat{E})$ that is structurally similar to G. We require that the output graph be over the same set of nodes as the original graph, that is, $\widehat{V} = V$. Thus, we focus on two graph-modification operations: addition and deletion of edges. Given two graphs $G(V, E)$ and $\widehat{G}(\widehat{V}, \widehat{E})$ defined over the same set of nodes ($\widehat{V} = V$) we measure their structural similarity using two metrics: the *degree anonymization cost* and the *structural cost*. The formal definitions of these metrics are given below.

Definition 3 For two graphs $G(V, E)$ and $\widehat{G}(\widehat{V}, \widehat{E})$ with degree sequences $\mathbf{d}$ and $\widehat{\mathbf{d}}$ defined over the same set of nodes ($V = \widehat{V}$), we define the degree anonymization cost between G and $\widehat{G}$ to be the L_1-norm of the difference of their degree sequences. That is,

$$\text{DA}\left(\widehat{\mathbf{d}}, \mathbf{d}\right) := L_1\left(\widehat{\mathbf{d}} - \mathbf{d}\right) := \sum_i \left|\widehat{\mathbf{d}}(i) - \mathbf{d}(i)\right|. \tag{14.1}$$

Definition 4 For two graphs $G(V, E)$ and $\widehat{G}(\widehat{V}, \widehat{E})$ defined over the same set of nodes ($V = \widehat{V}$), we define the structural difference between G and $\widehat{G}$ to be the symmetric difference of their sets of edges. That is,

$$\Delta\left(\widehat{G}, G\right) := \left|\widehat{E} \setminus E\right| + \left|E \setminus \widehat{E}\right|. \tag{14.2}$$

Given the above definitions we define the GRAPH ANONYMIZATION problem as follows.

Problem 1 (GRAPH ANONYMIZATION) Given a graph $G(V, E)$ with degree sequence $\mathbf{d}$, and an integer k, find a k-degree anonymous graph $\widehat{G}(V, \widehat{E})$ with degree sequence $\widehat{\mathbf{d}}$ such that (a) $\text{DA}\left(\widehat{\mathbf{d}}, \mathbf{d}\right)$ is minimized and (b) $\Delta\left(\widehat{G}, G\right)$ is also minimized.

Note that the above problem definition has two optimization objectives: it requires both the degree anonymization and the structural cost of the anonymization to be minimized. We first show that a solution to the GRAPH ANONYMIZATION problem that is optimal with respect to the one objective is not necessarily optimal with respect to the other. This is illustrated in the following observation.

Observation 1 *Consider graph $G(V, E)$ and $G'\left(V', E'\right)$ with $V = V' = \{w, x, y, z\}$, $E = \{(w, x), (y, z)\}$, and $E' = \{(w, z), (x, y)\}$. One can verify that $\text{DA}(\mathbf{d}_G, \mathbf{d}_{G'}) = 0$ while $\Delta\left(G, G'\right) = 4$. At the same time it is easy to construct another graph $G''\left(V'', E''\right)$ with $V'' = \{w, x, y, z\}$ and $E'' = \{(w, x), (y, x)\}$ such that $\Delta\left(G, G''\right) = 2 < \Delta\left(G, G'\right)$. At the same time $\text{DA}(\mathbf{d}_G, \mathbf{d}_{G''}) = 2$.*

One way of dealing with problems that have more than objective functions is to define the problem as a multiple-objective optimization problem, where the goal is to *simultaneously* optimize both the objectives. In this case, the situation can arise that two graphs $\widehat{G}$ and $\widehat{G}'$ are *incomparable*, e.g., $\text{DA}\left(\mathbf{d}_{\widehat{G}}, \mathbf{d}_G\right) < \text{DA}\left(\mathbf{d}_{\widehat{G}'}, \mathbf{d}_G\right)$, but $\Delta\left(\widehat{G}, G\right) > \Delta\left(\widehat{G}', G\right)$. In these cases, all one can hope for are *Pareto-optimal solutions* [5]. A graph $\widehat{G}$ is a Pareto-optimal solution if there does not exist another graph $\widehat{G}'$ that is at least as good as $\widehat{G}$ in both objectives and strictly better than $\widehat{G}$ in at least one of the two. The problem of finding all (approximate) Pareto-optimal solutions has been studied in [7].

In our case, we take a different approach: we prioritize our objectives. That is, we first focus on minimizing degree anonymization cost; then among all those graphs that have the same value of DA() cost, we try to pick the one with the minimum $\Delta()$ cost.

Note that the GRAPH ANONYMIZATION problem always has a *feasible* solution. For example, all edges not present in the input graph can be added. In this way, the graph becomes complete and all nodes have the same degree; thus, any degree anonymity requirement is satisfied (due to Proposition 1).

14.3.1 Restriction to Edge Additions

Problem 1 allows both for edge-addition and for edge-deletion modification operations. For the case where we focus our attention only on edge additions, the anonymized graph $\widehat{G}$ will be a *supergraph* of the original graph.

In this case minimizing $\text{DA}\left(\mathbf{d}_{\widehat{G}}, \mathbf{d}_G\right)$ is equivalent to minimizing $\Delta\left(\widehat{G}, G\right)$, because in the case of edge additions,

$$\begin{aligned}\Delta\left(\widehat{G}, G\right) &= \left|\widehat{E} \setminus E\right| = \left|\widehat{E}\right| - \left|E\right| \\ &= \frac{1}{2} L_1\left(\widehat{\mathbf{d}} - \mathbf{d}\right) = \frac{1}{2}\text{DA}\left(\mathbf{d}_{\widehat{G}}, \mathbf{d}_G\right).\end{aligned}$$

It is obvious that the case where only edge-deletion operations are allowed is equivalent, because edge deletions can be considered as edge additions in the complement of the input graph.

14.4 Overview of the Approach

We propose a two-step approach for the GRAPH ANONYMIZATION problem. For an input graph $G(V, E)$ with degree sequence $\mathbf{d}$ and an integer k, we proceed as follows:

1. First, starting from $\mathbf{d}$, we construct a new degree sequence $\widehat{\mathbf{d}}$ that is k-anonymous and such that the degree anonymization cost $\text{DA}(\widehat{\mathbf{d}}, \mathbf{d})$ is minimized.

2. Given the new degree sequence $\widehat{\mathbf{d}}$, we then construct a graph $\widehat{G}(V, \widehat{E})$ such that $\Delta\left(\widehat{G}, G\right)$ is minimized.

These two steps give rise to two problems, which we formally define and solve in subsequent sections. Performing step 1 translates into solving the DEGREE ANONYMIZATION problem defined as follows.

Problem 2 (DEGREE ANONYMIZATION) Given $\mathbf{d}$, the degree sequence of graph $G(V, E)$, and an integer k construct a k-anonymous sequence $\widehat{\mathbf{d}}$ such that $\text{DA}\left(\widehat{\mathbf{d}}, \mathbf{d}\right) = L_1(\widehat{\mathbf{d}} - \mathbf{d})$ is minimized.

Similarly, performing step 2 translates into solving the GRAPH CONSTRUCTION problem that we define below.

Problem 3 (GRAPH CONSTRUCTION) Given a graph $G(V, E)$ and a k-anonymous degree sequence $\widehat{\mathbf{d}}$, construct graph $\widehat{G}(V, \widehat{E})$ such that $\mathbf{d}_{\widehat{G}} = \widehat{\mathbf{d}}$ and $\Delta\left(\widehat{G}, G\right)$ is minimized.

In the next sections we develop algorithms for solving Problems 2 and 3.

14.5 Degree Anonymization

In this section we give algorithms for solving the DEGREE ANONYMIZATION problem. Given the degree sequence $\mathbf{d}$ of the original input graph $G(V, E)$, the algorithms output a k-anonymous degree sequence $\widehat{\mathbf{d}}$ such that the degree anonymization cost $\text{DA}(\mathbf{d}) = L_1(\widehat{\mathbf{d}} - \mathbf{d})$ is minimized.

We first give a dynamic-programming algorithm (DP) that solves the DEGREE ANONYMIZATION problem optimally in time $O(n^2)$. Then, we show how to modify it to achieve $O(nk)$ running time, and finally, for the restricted case of edge additions or edge deletions only, $O(n)$ time.

Given a (sorted) input degree sequence $\mathbf{d}$, let $\text{DA}(\mathbf{d}[1, i])$ be the degree anonymization cost of subsequence $\mathbf{d}[1, i]$. Additionally, let $I(\mathbf{d}[i, j])$ be the degree anonymization cost when all nodes $i, i+1, \ldots, j$ are put in the same anonymized group. Alternatively, this is the cost of assigning to all nodes $\{i, \ldots, j\}$ the same degree, d^*. That is,

$$I(\mathbf{d}[i, j]) = \sum_{\ell=i}^{j} |d^* - \mathbf{d}(\ell)|,$$

where d^* is the (integer) degree with property

$$d^* = \arg\min_{d} \sum_{\ell=i}^{j} |d - \mathbf{d}(\ell)|.$$

From [14] we know that d^* is the median of the values $\{\mathbf{d}(i), \ldots \mathbf{d}(j)\}$, and therefore given i and j, computing $I(\mathbf{d}[i, j])$ can be done optimally in $O(j - i)$ time (see [6] for details).

We now construct a set of dynamic-programming equations that solve the DEGREE ANONYMIZATION problem. For $i < 2k$, let

$$C(i) := I(\mathbf{d}[1, i]). \tag{14.3}$$

For $i \geq 2k$, let

$$C(i) := \min\left\{ I(\mathbf{d}[1, i]), \min_{k \leq t \leq i-k} \left\{ C(t) + I(\mathbf{d}[t+1, i]) \right\} \right\}. \tag{14.4}$$

When $i < 2k$, it is impossible to construct two different anonymized groups each of size k. As a result, the optimal degree anonymization of nodes $1, \ldots, i$ consists of a single group.

When $i \geq 2k$, the degree anonymization cost for the subsequence $\mathbf{d}[1, i]$ is the optimal degree anonymization cost of the subsequence $\mathbf{d}[1, t]$ plus the anonymization cost incurred by putting all nodes $t + 1, ..., i$ in the same group (provided this group is of size k or larger). The range of variable t, as defined in (14.4), is restricted so that all groups examined, including the first and last ones, are of size at least k.

Running time of the DP *algorithm.* For an input degree sequence of size n, the running time of the DP algorithm that implements Recursions (14.3) and (14.4) is $O(n^2)$; first, the values of $I(\mathbf{d}[i, j])$ for all $i < j$ can be computed in an $O(n^2)$ preprocessing step. Then, for every i the algorithm goes through at most $n - 2k + 1$ different values of t for evaluating Recursion (14.4). Since there are n different values of i, the total running time is $O(n^2)$.

In fact the running time of the DP algorithm can further improve from $O(n^2)$ to $O(nk)$. The core idea for this speedup lies in the following simple observation: *no anonymous group should be of size larger than* $2k-1$. *If any group is larger than or equal to* $2k$, *it can be broken into two subgroups with equal or lower overall degree anonymization cost*. Using this observation, the preprocessing step that computes the values of $I(\mathbf{d}[i, j])$, does not have to consider all the combinations of (i, j) pairs, but for every i consider only j's such that $k \leq j - i + 1 \leq 2k - 1$. Thus, the running time for this step drops to $O(nk)$.

Similarly, for every i, we do not have to consider all t's in the range $k \leq t \leq i-k$ as in Recursion (14.4), but only t's in the range $\max\{k, i - 2k + 1\} \leq t \leq i - k$. Therefore, Recursion (14.4) can be replaced by

$$C(i) := \min_{t \in S_i} \left\{ C(t) + I(\mathbf{d}[t+1, i]) \right\}, \tag{14.5}$$

where

$$S_i := \{t \mid \max\{k, i - 2k + 1\} \leq t \leq i - k\}.$$

For this range of values of t we guarantee that the first group has size at least k, and the last one has size between k and $2k-1$. Therefore, for every i the algorithm

goes through at most k different values of t for evaluating the new recursion. Since there are $O(n)$ different values of i, the overall running time of the DP algorithm is $O(nk)$.

Therefore, we have the following.

Theorem 1 *Problem 2 can be solved in polynomial time using the* DP *algorithm described above. The running time of the* DP *algorithm when the degrees of the nodes can either increase or decrease is* $O(nk)$.

14.5.1 Restriction to Edge Additions

Again, if we restrict our attention to Problem 1 where only edge additions are allowed, then the degrees of the nodes can only increase in the DEGREE ANONYMIZATION problem. That is, if $\mathbf{d}$ is the original sequence and $\widehat{\mathbf{d}}$ is the k-anonymous degree sequence, then for every $1 \leq i \leq n$ we have that $\widehat{\mathbf{d}}(i) \geq \mathbf{d}(i)$. In this case, the DP algorithm described in the previous section can solve the DEGREE ANONYMIZATION problem. The only difference is in the evaluation of cost $I(\mathbf{d}[i, j])$ that corresponds to the L_1 cost of putting all nodes $i, i+1, \ldots, j$ in the same anonymized group. Note that the indices correspond to the ordering of the nodes in nonincreasing order of their degree in $\mathbf{d}$. Therefore, if the degrees of the nodes can only increase, every group will be assigned the degree of the node with the highest degree. That is,

$$I\left(\mathbf{d}\left[i, j\right]\right) = \sum_{\ell=i}^{j} (\mathbf{d}(i) - \mathbf{d}(\ell)). \tag{14.6}$$

In this case, the running time of the DP algorithm can be further improved as follows.

Letting $f_i(t) := C(t) + I(\mathbf{d}[t+1, i])$, Recursion (14.5) is

$$C(i) := \min_{t \in S_i} f_i(t).$$

For given i, if $f_i(s) \geq f_i(t)$ for all $t \in S_i$ with $t > s$, then plainly $f_i(s)$ is greater than or equal to the minimum of $f_i(t)$, taken over *all* $t \in S_i$. So it is enough to have the values of $f_i(s)$, for $s \in S_i$ such that $f_i(s) < f_i(t)$ for all $t \in S_i$ with $t > s$. Consider now such s and the next largest such s'. We have $f_i(s') - f_i(s) > 0$, which suggests $f_{i+1}(s') - f_{i+1}(s) > 0$, but is this true, and what happens as i increases further? The next lemma delimits the possibilities.

Lemma 1 *For fixed* s *and* s' *with* $s' > s$, *let* $F(i) := f_i(s') - f_i(s)$. *Then*

$$F(i+1) - F(i) = \mathbf{d}(s'+1) - \mathbf{d}(s+1) \leq 0.$$

This implies the following.

- *Suppose* $\mathbf{d}(s'+1) = \mathbf{d}(s+1)$. *Then* $F(i') = F(i)$ *for all* $i' \geq i$.
- *Suppose* $\mathbf{d}(s'+1) < \mathbf{d}(s+1)$. *Then* $F(i) \leq 0$ *implies* $F(i') \leq 0$ *for all* $i' \geq i$. *If* $F(i) > 0$, *then* $F(i') > 0$ *for* i' *with* $i' < i + F(i)/(\mathbf{d}(s+1) - \mathbf{d}(s'+1))$, *and* $F(i') \leq 0$ *for* $i' \geq i + F(i)/(\mathbf{d}(s+1) - \mathbf{d}(s'+1))$.

Proof The expression for $F(i+1) - F(i)$ follows from expanding out the definitions and manipulating as follows. We have

$$\begin{aligned} f_{i+1}(s) - f_i(s) &= C(\mathbf{d}[1,s]) + I(\mathbf{d}[s+1,i+1]) - [C(\mathbf{d}[1,s]) + I(\mathbf{d}[s+1,i])] \\ &= I(\mathbf{d}[s+1,i+1]) - I(\mathbf{d}[s+1,i]) \\ &= \sum_{s+1 \leq \ell \leq i+1} (\mathbf{d}(s+1) - \mathbf{d}(\ell)) - \sum_{s+1 \leq \ell \leq i} (\mathbf{d}(s+1) - \mathbf{d}(\ell)) \\ &= \mathbf{d}(s+1) - \mathbf{d}(i+1), \end{aligned}$$

and so

$$\begin{aligned} F(i+1) - F(i) &= f_{i+1}(s') - f_{i+1}(s) - (f_i(s') - f_i(s)) \\ &= f_{i+1}(s') - f_i(s') - (f_{i+1}(s) - f_i(s)) \\ &= \mathbf{d}(s'+1) - \mathbf{d}(i+1) - (\mathbf{d}(s+1) - \mathbf{d}(i+1)) \\ &= \mathbf{d}(s'+1) - \mathbf{d}(s+1), \end{aligned}$$

which proves the first claim. From this it follows immediately that $F(i') - F(i) = (i'-i)(\mathbf{d}(s'+1) - \mathbf{d}(s+1))$, implying the remaining claims. □

The discussion just before the lemma says that to compute $\min_{t \in S_i} f_i(t)$ for increasing values of i, it is enough to be able to maintain, as i goes from 1 to n, the list of indices

$$L_i := \{s \in S_i \mid f_i(s) < \min_{s < t \in S_i} f_i(t)\}. \tag{14.7}$$

Specifically, L_i is represented as a doubly linked list. How does this list change as i increases? If $s \notin L_i$, then $f_i(s) \geq f_i(t)$ for some $t > s$, and so by the lemma, $f_{i'}(s) \geq f_{i'}(t)$ for $i' \geq i$; that is, if s is not included now, it will not be included later.

If L_i has been maintained up to time i, then its first entry is sufficient to allow the computation of (14.4): for given i and $s < i$, the value of $f_i(s)$ can be computed in constant time, using previously computed values stored in array C and the prefix sums $\sum_{\ell < i} \mathbf{d}(\ell)$.

To maintain the list, we will consider the "life cycle" of an entry in it. Such an entry s is "born" at time $i = s + k$, when $i - k$ is added to the end of L_i. An entry can "die," that is, no longer qualify to be in L_i, in a few ways.

An entry can die when it gets too old, namely, when $s < \max\{k, i-2k+1\}$, and so is no longer in S_i.

An entry can also die when a new entry is added whose f value is less than or equal to its f value; here if $s \in L_i$ dies, all $t > s$ in L_i must also die, since they have larger $f_i(t)$ values. Thus when $i-k$ is added, the entries $s \in L_i$ can be examined, in decreasing order, to check if $f_i(i-k) < f_i(s)$; as such s are found; they are deleted from L_i.

We will also make an entry die in one other way: if $f_i(s)$ and $f_i(s')$ are consecutive entries in L_i, so that $s < s'$ and $f_i(s) < f_i(s')$, then from the lemma, there is a future time $i' = D(s, s')$ at which $f_{i'}(s) \geq f_{i'}(s')$, so that s should die. We will maintain that, for each consecutive pair s, s' of entries in L_i, there is a "death notice" for s, at time $D(s, s')$. At that time, when the "death notice" is processed, the entry for s is removed from L_i (if it has not already died), and a death notice is added for the former neighbor $s'' < s$, whose new rightward neighbor is the former rightward neighbor of s. The death notice $D(s, s')$ includes a pointer to the list node for s, which has a backpointer to the death notice, so that if the node for s is removed for other reasons, the death notice is removed also.

When an entry dies, $O(1)$ work is done for it, including the generation of at most one new death notice. The processing of that death notice, in the future, requires $O(1)$ work, so the life cycle of a node requires $O(1)$ work.

We have proven the following theorem.

Theorem 2 *Problem 2 for the restricted case of edge additions (or deletions) operations, where the degrees of the nodes can only increase (or decrease) can be solved optimally using the* DP *algorithm described above in time* $O(n)$.

14.6 Graph Construction

In this section we present algorithms for solving the GRAPH CONSTRUCTION problem. Given the original graph $G(V, E)$ and the desired k-anonymous degree sequence $\widehat{\mathbf{d}}$ output by the DP algorithm, we construct a k-degree anonymous graph $\widehat{G}(V, \widehat{E})$ such that $\Delta\left(\widehat{G}, G\right)$ is minimized.

14.6.1 Basics on Realizability of Degree Sequences

Before giving the actual algorithms for the GRAPH CONSTRUCTION problem, we first present some known facts about the realizability of degree sequences for simple graphs. Later on, we extend some of these results in our own problem setting.

Definition 5 A degree sequence $\mathbf{d}$, with $\mathbf{d}(1) \geq, \cdots, \geq \mathbf{d}(n)$ is called *realizable* if and only if there exists a simple graph whose nodes have precisely this sequence of degrees.

Erdös and Gallai [8] have stated the following *necessary* and *sufficient* condition for a degree sequence to be realizable.

Lemma 2 ([8]) *A degree sequence* $\mathbf{d}$ *with* $\mathbf{d}(1) \geq \cdots \geq \mathbf{d}(n)$ *and* $\sum_i \mathbf{d}(i)$ *even is realizable if and only if for every* $1 \leq \ell \leq n-1$ *it holds that*

$$\sum_{i=1}^{\ell} \mathbf{d}(i) \leq \ell(\ell-1) + \sum_{i=\ell+1}^{n} \min\{\ell, \mathbf{d}(i)\} \tag{14.8}$$

Informally, Lemma 2 states that for each subset of the ℓ highest-degree nodes, the degrees of these nodes can be "absorbed" within the nodes and the outside degrees. The proof of Lemma 2 is inductive [10] and it provides a natural construction algorithm, which we call `ConstructGraph` (see Algorithm 1 for the pseudocode).

The `ConstructGraph` algorithm takes as input the desired degree sequence $\mathbf{d}$ and outputs a graph with exactly this degree sequence, if such graph exists. Otherwise it outputs a "No" if such graph does not exist. The algorithm is iterative and in each step it maintains the residual degrees of vertices. In each iteration it picks an arbitrary node v and adds edges from v to $\mathbf{d}(v)$ nodes of *highest* residual degree, where $\mathbf{d}(v)$ is the residual degree of v. The residual degrees of these $\mathbf{d}(v)$ nodes are decreased by one. If the algorithm terminates and outputs a graph, then this graph has the desired degree sequence. If at some point the algorithm cannot make the required number of connections for a specific node, then it outputs "No" meaning that the input degree sequence is not realizable.

Note that the `ConstructGraph` algorithm is an *oracle* for the realizability of a given degree sequence; if the algorithm outputs "No", then this means that there does not exist a simple graph with the desired degree sequence.

Algorithm 1 The `ConstructGraph` algorithm

Input: A degree sequence $\mathbf{d}$ of length n.
Output: A graph $G(V, E)$ with nodes having degree sequence $\mathbf{d}$ or "No" if the input sequence is not realizable.

$V \leftarrow \{1, \ldots, n\}$, $E \leftarrow \emptyset$
if $\sum_i \mathbf{d}(i)$ is odd **then**
  Halt and return "No"
while 1 **do**
  if there exists $\mathbf{d}(i)$ such that $\mathbf{d}(i) < 0$ **then**
    Halt and return "No"
  if the sequence $\mathbf{d}$ are all zeros **then**
    Halt and return $G(V, E)$
  Pick a random node v with $\mathbf{d}(v) > 0$
  $\mathbf{d}(v) \leftarrow 0$
  $V_{\mathbf{d}(v)} \leftarrow$ the $\mathbf{d}(v)$-highest entries in $\mathbf{d}$ (other than v)
  for each node $w \in V_{\mathbf{d}(v)}$ **do**
    $E \leftarrow E \cup (v, w)$
    $\mathbf{d}(w) \leftarrow \mathbf{d}(w) - 1$

Running time of the `ConstructGraph` *algorithm:* If n is the number of nodes in the graph and $d_{\max} = \max_i \mathbf{d}(i)$, then the running time of the `ConstructGraph` algorithm is $O(nd_{\max})$. This running time can be achieved by keeping an array A of size

$d_{\max}$ such that $A[\mathbf{d}(i)]$ keeps a hash table of all the nodes of degree $\mathbf{d}(i)$. Updates to this array (degree changes and node deletions) can be done in constant time. For every node i at most $d_{\max}$ constant-time operations are required. Since there are n nodes the running time of the algorithm is $O(nd_{\max})$. In the worst case, $d_{\max}$ can be of order $O(n)$, and in this case the running time of the `ConstructGraph` algorithm is quadratic. In practice, $d_{\max}$ is much less than n, which makes the algorithm very efficient in practical settings.

Note that the random node in Step 9 of Algorithm 1 can be replaced by either the current highest-degree node or the current lowest-degree node. When we start with higher degree nodes, we get topologies that have very dense cores, while when starting with lower degree nodes, we get topologies with very sparse cores. A random pick is a balance between the two extremes. The running time is not affected by this choice, due to the data structure A.

14.6.2 The `Greedy_Swap` *Algorithm*

Let $\widehat{\mathbf{d}}$ be a k-anonymous degree sequence output by `DP` algorithm. Let us additionally assume for now that $\widehat{\mathbf{d}}$ is realizable so that the `ConstructGraph` algorithm with input $\widehat{\mathbf{d}}$ outputs a simple graph $\widehat{G}_0(V, \widehat{E}_0)$ with degree sequence exactly $\widehat{\mathbf{d}}$. Although $\widehat{G}_0$ is k-degree anonymous, its structure may be quite different from the original graph $G(V, E)$. The `Greedy_Swap` algorithm is a greedy heuristic that given $\widehat{G}_0$ and G, it transforms $\widehat{G}_0$ into $\widehat{G}(V, \widehat{E})$ with degree sequence $\mathbf{d}_{\widehat{G}} = \widehat{\mathbf{d}} = \mathbf{d}_{\widehat{G}_0}$ such that $\Delta\left(\widehat{G}, G\right)$ is minimized.

At every step i, the graph $\widehat{G}_{i-1}(V, \widehat{E}_{i-1})$ is transformed into the graph $\widehat{G}_i(V, E_i)$ such that $\widehat{\mathbf{d}}_{\widehat{G}_0} = \widehat{\mathbf{d}}_{\widehat{G}_{i-1}} = \widehat{\mathbf{d}}_{\widehat{G}_i} = \widehat{\mathbf{d}}$ and $\Delta\left(\widehat{G}_i, G\right) < \Delta\left(\widehat{G}_{i-1}, G\right)$. The transformation is made using *valid swap* operations defined as follows:

Definition 6 Consider a graph $\widehat{G}_i(V, \widehat{E}_i)$. A valid swap operation is defined by four vertices i, j, k, and l of $\widehat{G}_i(V, \widehat{E}_i)$ such that $(i, k) \in \widehat{E}_i$ and $(j, l) \in \widehat{E}_i$ and $(i, j) \notin \widehat{E}_i$ and $(k, l) \notin \widehat{E}_i$ or $(i, l) \notin \widehat{E}_i$ and $(j, k) \notin \widehat{E}_i$. A valid swap operation transforms $\widehat{G}_i$ to $\widehat{G}_{i+1}$ by updating the edges as follows:

$$\widehat{E}_{i+1} \leftarrow \widehat{E}_i \setminus \{(i,k),(j,l)\} \cup \{(i,j),(k,l)\}, \text{ or}$$
$$\widehat{E}_{i+1} \leftarrow \widehat{E}_i \setminus \{(i,k),(j,l)\} \cup \{(i,l),(j,k)\}.$$

A visual illustration of the swap operation is shown in Fig. 14.2. It is clear that performing valid swaps on a graph leaves the degree sequence of the graph intact. The pseudocode for the `Greedy_Swap` algorithm is given in Algorithm 2. At each iteration of the algorithm, the swappable pair of edges e_1 and e_2 is picked to be swapped to edges e_1' and e_2'. Swaps that can mostly reduce the structural difference (the Δ () function) between the original graph and the new graph are chosen at each iteration. The `Greedy_Swap` algorithm halts when there are no more valid swaps that can decrease the Δ () function.

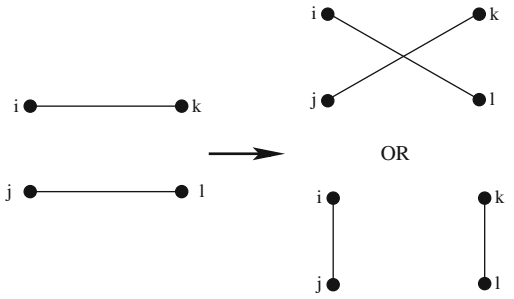


Fig. 14.2 The swap transformation

Algorithm 2 The `Greedy_Swap` algorithm

Input: An initial graph $\widehat{G}_0(V, \widehat{E}_0)$ and the input graph $G(V, E)$.
Output: Graph $\widehat{G}(V, \widehat{E})$ with the same degree sequence as $\widehat{G}_0$, such that $\Delta\left(\widehat{G}, G\right)$ is minimized.

$\widehat{G}(V, \widehat{E}) \leftarrow \widehat{G}_0(V, \widehat{E}_0)$
$\left(c, (e_1, e_2, e'_1, e'_2)\right) =$ `Find_Best_Swap`$(\widehat{G})$
while $c > 0$ **do**
  $\widehat{E} = \widehat{E} \setminus \{e_1, e_2\} \cup \{e'_1, e'_2\}$
  $\left(c, (e_1, e_2, e'_1, e'_2)\right) =$ `Find_Best_Swap`
return $\widehat{G}$

Algorithm 3 An overall algorithm for solving the GRAPH CONSTRUCTION problem; the realizable case

Input: A realizable degree sequence $\widehat{\mathbf{d}}$ of length n.
Output: A graph $\widehat{G}(V, E')$ with degree sequence $\widehat{\mathbf{d}}$ and $E \cap E' \approx E$.

$\widehat{G}_0 =$ `ConstructGraph`$(\widehat{\mathbf{d}})$
$\widehat{G} =$ `Greedy_Swap`$(\widehat{G}_0)$

Algorithm 3 gives the pseudocode of the whole process of solving the GRAPH CONSTRUCTION problem when the degree sequence $\widehat{\mathbf{d}}$ is realizable. The first step involves a call to the `ConstructGraph` algorithm, which we have described in Section 14.6.1, Algorithm 1. The `ConstructGraph` algorithm will return a graph $\widehat{G}_0$ with degree distribution $\widehat{\mathbf{d}}$. The `Greedy_Swap` algorithm then takes graph $\widehat{G}_0$ as input, and then outputs a k-degree anonymous graph that has degree sequence $\widehat{\mathbf{d}}$ and large overlap in its set of edges with the original graph.

A naive implementation of the algorithm would require time $O(I|\widehat{E}_0|^2)$, where I is the number of iterations of the greedy step and $|\widehat{E}_0|$ the number of edges in the input graph. Given that $|\widehat{E}_0| = O(n^2)$, the running time of the `Greedy_Swap` algorithm could be $O(n^4)$, which is daunting for large graphs. However, we employ a simple sampling procedure that considerably improves the running time. Instead of doing the greedy search over the set of all possible edges, we uniformly at random pick a subset of size $O(\log|\widehat{E}_0|) = O(\log n)$ of the edges and run the algorithm on those. This reduces the running time of the greedy algorithm to $O(I \log^2 n)$, which makes it efficient even for very large graphs. As we show in our experimental

evaluation, the Greedy_Swap algorithm performs very well in practice, even in cases where it starts with graph $\widehat{G}_0$ that shares small number of edges with G.

14.6.2.1 The Probing Scheme

In the discussion above we have assumed that the degree sequence input in the ConstructGraph algorithm is realizable. However, it might well be the case that the ConstructGraph algorithm outputs a "No," i.e., there does not exist a graph with the required degree sequence. In this case we invoke a Probing scheme described below. The Probing scheme is a randomized iterative process that tries to slightly change the degree sequence $\widehat{\mathbf{d}}$. The pseudocode of the Probing scheme is shown in Algorithm 4.

Algorithm 4 The Probing scheme

Input: Input graph $G(V, E)$ with degree distribution $\mathbf{d}$ and integer k.
Output: Graph $\widehat{G}(V, \widehat{E})$ with k-anonymous degree sequence $\widehat{\mathbf{d}}$.

1: $\widehat{\mathbf{d}} = \text{DP}(\mathbf{d})$
2: $\left(\text{realizable}, \widehat{G}\right) = \text{ConstructGraph}(\widehat{\mathbf{d}})$
3: **while** realizable = "No" **do**
4: $\quad \mathbf{d} = \mathbf{d} + \text{random\_noise}$
5: $\quad \widehat{\mathbf{d}} = \text{DP}(\mathbf{d})$
6: $\quad \left(\text{realizable}, \widehat{G}\right) = \text{ConstructGraph}(\widehat{\mathbf{d}})$
7: Return $\widehat{G}$

For input graph $G(V, E)$ and integer k, the Probing scheme first constructs the k-anonymous sequence $\widehat{\mathbf{d}}$ by invoking the DP algorithm. If the subsequent call to the ConstructGraph algorithm returns a graph $\widehat{G}$, then Probing outputs this graph and halts. If ConstructGraph returns "No," then Probing slightly increases some of the entries in $\mathbf{d}$ via the addition of uniform noise – the specifics of the noise-addition strategy is further discussed in the next paragraph. The new noisy version of $\mathbf{d}$ is then fed as input to the DP algorithm again. A new version of the $\widehat{\mathbf{d}}$ is thus constructed and input to the ConstructGraph algorithm to be checked. The process of noise addition and checking is repeated until a graph is output by ConstructGraph. Note that this process will always terminate because in the worst case, the noisy version of $\mathbf{d}$ will contain all entries equal to $n - 1$, and there exists a complete graph that satisfies this sequence and is k-degree anonymous.

Since the Probing procedure will always terminate, the key question is how many times the **while** loop is executed. This depends, to a large extent, on the noise-addition strategy. In our implementation, we examine the nodes in increasing order of their degrees and slightly increase the degree of a single node in each iteration. This strategy is suggested by the degree sequences of the input graphs. In most of these graphs there is a small number of nodes with very high degrees. However, rarely any two of these high-degree nodes share exactly the same degree. In fact, we often observe big differences among them. On the contrary, in most graphs there

is a large number of nodes with the same small degrees (close to 1). Given such a graph, the DP algorithm will be forced to increase the degrees of some of the large-degree nodes a lot, while leaving the degrees of small-degree nodes untouched. In the anonymized sequence thus constructed, a small number of high-degree nodes will need a large number of nodes to connect their newly added edges. However, since the degrees of small-degree nodes do not change in the anonymized sequence, the demand of edge end-points imposed by the high-degree nodes cannot be facilitated. Therefore, by slightly increasing the degrees of small-degree nodes in $\mathbf{d}$ we force the DP algorithm to assign them higher degrees in the anonymized sequence $\widehat{\mathbf{d}}$. In that way, there are more additional free edges end-points to connect with the anonymized high-degree nodes.

From our experiments on a large spectrum of synthetic and real-world data, we observe that, in most cases, the extra edge additions incurred by the Probing procedure are negligible; that is, the degree sequences produced by the DP are almost realizable and, more importantly, realizable with respect to the input graph G. Therefore, the Probing is rarely invoked, and even if it is invoked, only a very small number of repetitions are needed. We further discuss this in the experimental section of this chapter.

14.6.3 Restriction to Edge Additions

In this section we give yet another graph construction algorithm for the case where only edge additions are allowed to the input graph. In this case, not only do we need to construct a graph $\widehat{G}$ with a given degree sequence $\widehat{\mathbf{d}}$, but we also require that $E \subseteq \widehat{E}$. We capture these two requirements in the following definition of *realizability of* $\widehat{\mathbf{d}}$ *subject to graph* G.

Definition 7 Given input graph $G(V, E)$, we say that degree sequence $\widehat{\mathbf{d}}$ is realizable subject to G, if and only if there exists a simple graph $\widehat{G}(V, \widehat{E})$ whose nodes have precisely the degrees suggested by $\widehat{\mathbf{d}}$ and $E \subseteq \widehat{E}$.

Given the above definition we have the following alternation of Lemma 2.

Lemma 3 *Consider degree sequence* $\widehat{\mathbf{d}}$ *and graph* $G(V, E)$ *with degree sequence* $\mathbf{d}$. *Let vector* $\mathbf{a} = \widehat{\mathbf{d}} - \mathbf{d}$ *such that* $\sum_i \mathbf{a}(i)$ *is even. If* $\widehat{\mathbf{d}}$ *is realizable subject to graph* G *then*

$$\sum_{i \in V_\ell} \mathbf{a}(i) \leq \sum_{i \in V_\ell} \left(\ell - 1 - \mathbf{d}^\ell(i)\right) + \sum_{i \in V - V_\ell} \min\{\ell - \mathbf{d}^\ell(i), \mathbf{a}(i)\}, \tag{14.9}$$

where $\mathbf{d}^\ell(i)$ *is the degree of node* i *in the input graph* G *when counting only edges in* G *that connect node* i *to one of the nodes in* V_ℓ. *Here* V_ℓ *is an ordered set of* ℓ

nodes with the ℓ largest $\mathbf{a}(i)$ *values sorted in decreasing order. In other words, for every pair of nodes* (u, v) *where* $u \in V_\ell$ *and* $v \in V \setminus V_\ell$*, it holds that* $\mathbf{a}(u) \geq \mathbf{a}(v)$ *and* $|V_\ell| = \ell$.

Although the proof of the lemma is omitted due to space constraints, one can see the similarity between inequalities (14.8) and (14.9); if G is a graph with no edges between its nodes, then $\mathbf{a}$ is the same as $\widehat{\mathbf{d}}$, $\mathbf{d}^\ell(i)$ is zero, and the two inequalities become identical.

Lemma 3 states that inequality (14.9) is just a *necessary* condition for realizability subject to the input graph G. Thus, if inequality (14.9) does not hold, we can conclude that for input graph $G(V, E)$, there does not exist a graph $\widehat{G}(V, \widehat{E})$ with degree sequence $\widehat{\mathbf{d}}$ such that $E \subseteq \widehat{E}$.

Although Lemma 3 gives only a necessary condition for realizability subject to an input graph G, we still want to devise an algorithm for constructing a degree anonymous graph $\widehat{G}$, a supergraph of G, if such a graph exists. We call this algorithm the `Supergraph`, which is an extension of the `ConstructGraph` algorithm. (We omit the pseudocode of `Supergraph` due to space limits.)

The inputs to the `Supergraph` are the original graph G and the desired k-anonymous degree distribution $\widehat{\mathbf{d}}$. The algorithm operates on the sequence of *additional degrees* $\mathbf{a} = \widehat{\mathbf{d}} - \mathbf{d}_G$ in a manner similar to the one the `ConstructGraph` algorithm operates on the degrees $\mathbf{d}$. However, since $\widehat{G}$ is drawn on top of the original graph G, we have the additional constraint that edges already in G cannot be drawn again.

The `Supergraph` first checks whether inequality (14.9) is satisfied and returns "No" if it does not. Otherwise it proceeds iteratively and in each step it maintains the residual additional degrees $\mathbf{a}$ of the vertices. In each iteration it picks an arbitrary vertex v and adds edges from v to $\mathbf{a}(v)$ vertices of *highest* residual additional degree, ignoring nodes v' that are already connected to v in G. For every new edge (v, v'), $\mathbf{a}(v')$ is decreased by 1. If the algorithm terminates and outputs a graph, then this graph has degree sequence $\widehat{\mathbf{d}}$ and is a supergraph of the original graph. If the algorithm does not terminate, then it outputs "Unknown," meaning that there might exist a graph, but the algorithm is unable to find it. Though `Supergraph` is similar to `ConstructGraph`, it *is not* an oracle; that is, if the algorithm does not return a graph $\widehat{G}$ supergraph of G, it does not necessarily mean that such a graph does not exist.

For degree sequences of length n and $a_{\max} = \max_i \mathbf{a}(i)$ the running time of the `Supergraph` algorithm is $O(na_{\max})$, using the same data structures as those described in Section 14.6.1.

14.7 Experiments

In this section we evaluate the performance of the proposed graph anonymization algorithms.

14.7.1 Data Sets

We use both synthetic and real-world data sets. For the experiments with synthetic data sets, we generate *random*, *small-world*, and *scale-free* graphs.

Random graphs: Random graphs are graphs with nodes randomly connected to each other with probability p. Given the number of nodes n and the parameter p, a random graph is generated by creating an edge between each pair of nodes u and v with probability p. We use $\mathcal{G}_R$ to denote the family of graphs generated by this data-generation model and G_R to denote a member of the family.

Small-world graphs: A small-world graph is a type of graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops. This kind of graphs has large *clustering coefficient* (CC) that is significantly higher than expected by random chance and small *average path length* (APL) that is close to that of an equivalent random graph. The average path length is defined as the average length of the shortest path between all pairs of reachable nodes. The clustering coefficient is defined as the average fraction of pairs of neighbors of a node that are also connected to each other. These two indices, along with the degree distribution, are considered as standard measures in graph-analysis studies. We generate small-world graphs using the model proposed in [20]. We denote by $\mathcal{G}_W$ the family of graphs generated by this model and G_W the members of this family. The data-generation process is controlled by a parameter α that determines the extent to which the graph exhibits community structure. Values of α in the range $[5, 7]$ generate small-world graphs. We have additionally conducted experiments with small-world graphs generated using the alternative model proposed in [21]. However, since the results we obtained are very similar to the results obtained by using graphs in $\mathcal{G}_W$, we do not report them here due to space limitations.

Scale-free graphs: The scale-free graphs correspond to graphs with power-law degree distribution. In a power-law graph the probability that a node has degree d is proportional to $d^{-\gamma}$. The power-law distribution is determined by the exponent γ. The value of γ may vary, taking values between 2 and 3 for most real networks. We use the model proposed by Barabási and Albert [3] to generate scale-free graphs. The graph-generation process proceeds by inserting nodes sequentially. Each new node is initially connected to ℓ already existing nodes with probability proportional to their degree. We use $\mathcal{G}_{BS}$ to denote the family of graphs generated by this model and G_{BS} to denote members of the family.

For the real-world data, we use the *enron*, the *powergrid*, and the *co-authors* graphs.

Enron graph: The Enron email graph (available at http://www.cs.cmu.edu/enron/) is derived from a corpus of emails sent to and from managers at Enron Corporation. This data was originally made public by the Federal Energy Regulatory Commission. The data set contains 151 users. An edge between two users is added if they have corresponded at least five times.

Powergrid graph: In this graph, the nodes represent generators, transformers, and substations in a powergrid network; the edges represent high-voltage transmission lines between them. The data set is available at http://www.cs.helsinki.fi/u/tsaparas/MACN2006/.

Co-authors graph: The co-authors data set consists of 7955 authors of papers in database and theory conferences and it is available at the collection of Computer Science Bibliographies at http://liinwww.ira.uka.de/bibliography/. The co-authors graph is constructed by creating undirected edges between authors that have co-authored paper.

Table 14.1 summarizes the properties of the graphs we used for our experiments. All the graphs are simple, unweighted, and undirected.

Table 14.1 Structural properties of the graphs used for the experiments

	#Nodes	#Edges	APL	CC
$\mathcal{G}_W$ ($\alpha = 6$)	1000	5000	9.15	0.77
$\mathcal{G}_R$	1000	5000	3.27	0.01
$\mathcal{G}_{BS}$ ($\gamma = 3$)	1000	2995	3.57	0.02
enron	151	502	3.32	0.46
powergrid	4941	6594	9.12	0.10
co-authors	7955	10,055	6.00	0.64

14.7.2 Evaluating GRAPH CONSTRUCTION *Algorithms*

In this section we evaluate the performance of `Greedy_Swap`, `Greedy_Swap_Additions`, and `Supergraph` algorithms. Since the k-degree anonymity is guaranteed by construction, we only need to look at the structural similarity between the input and output graphs. We use a set of evaluation measures listed below and we report our results for different synthetic and real-world graphs.

Anonymization cost $L_1(\mathbf{d}_A - \mathbf{d})$
This is the L_1 norm of the vector of differences between the k-anonymous degree sequence obtained using algorithm $Algo \in$ {`Greedy_Swap`, `Greedy_Swap_Additions`, `Supergraph`} and the degree sequence of the original graph. The smaller the value of $L_1(\mathbf{d}_A - \mathbf{d})$ the better the qualitative performance of the algorithm. Figures 14.3–14.6 summarize the anonymization cost of the different algorithms as a function of $k = \{5, 10, 15, 20, 25, 50, 100\}$ for synthetic data sets $G_W \in \mathcal{G}_W$ with $\alpha = 6$, $G_{BS} \in \mathcal{G}_{BS}$, and *powergrid* and *co-authors* data. From

the plots, we can observe that `Greedy_Swap` always has the lowest anonymization cost. This is because `Greedy_Swap` allows the degrees of nodes to either increase or decrease. On the other hand, `Greedy_Swap_Additions` and `Supergraph` have relatively higher cost due to their increase-only nature.

If all the anonymized degree sequences are realizable, the optimal cost for both `Greedy_Swap_Additions` and `Supergraph` should be the same. However, we note that in the case of $\mathcal{G}_{BS}$ graphs, $L_1(\mathbf{d}_{\texttt{Supergraph}} - \mathbf{d})$ cost is relatively high for all values of k (see Fig. 14.4). This is due to two reasons: (1) The degrees of graphs in $\mathcal{G}_{BS}$ have a power-law distribution. This causes large differences among the degrees of high-degree nodes, meaning that the degrees of high-degree nodes have to be changed significantly in order to meet the degree anonymous requirement. (2) The `Supergraph` algorithm constructs the degree anonymous graph by extending the input graph, and it is the only of our proposed algorithms that tries to comply with all the edge constraints imposed by the input graph. Therefore, it can potentially add more noise (in the `Probing` phase) than other algorithms that build the graph from scratch.

Clustering coefficient (CC)

We compare the clustering coefficients of the anonymized graphs with the clustering coefficients of the original graphs. Figures 14.3–14.6 summarize our findings. In all

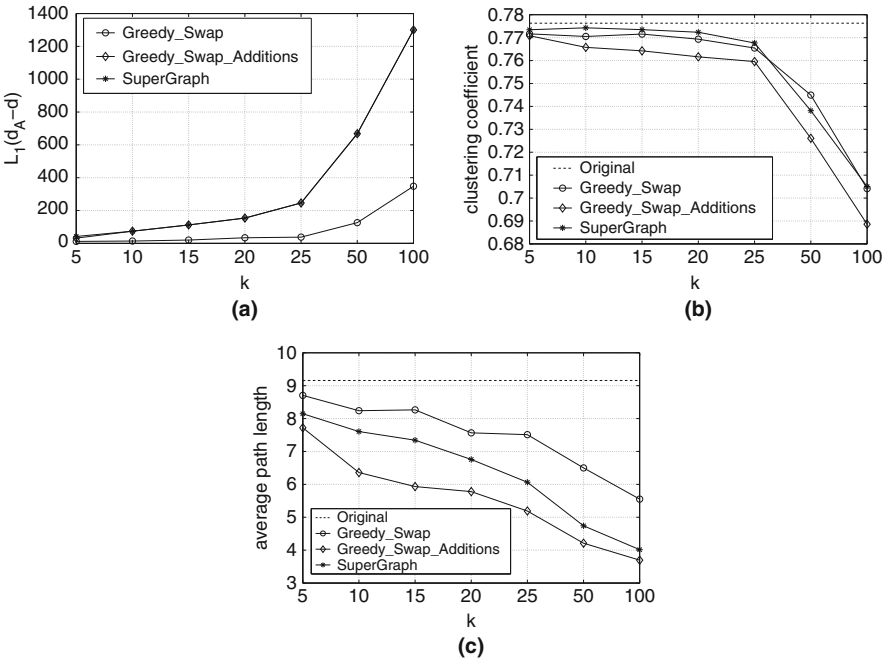


Fig. 14.3 Synthetic data sets: small-world graphs $G_W \in \mathcal{G}_W$, with $\alpha = 6$. (**a**) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k (**b**) CC as a function of k (**c**) APL as a function of k

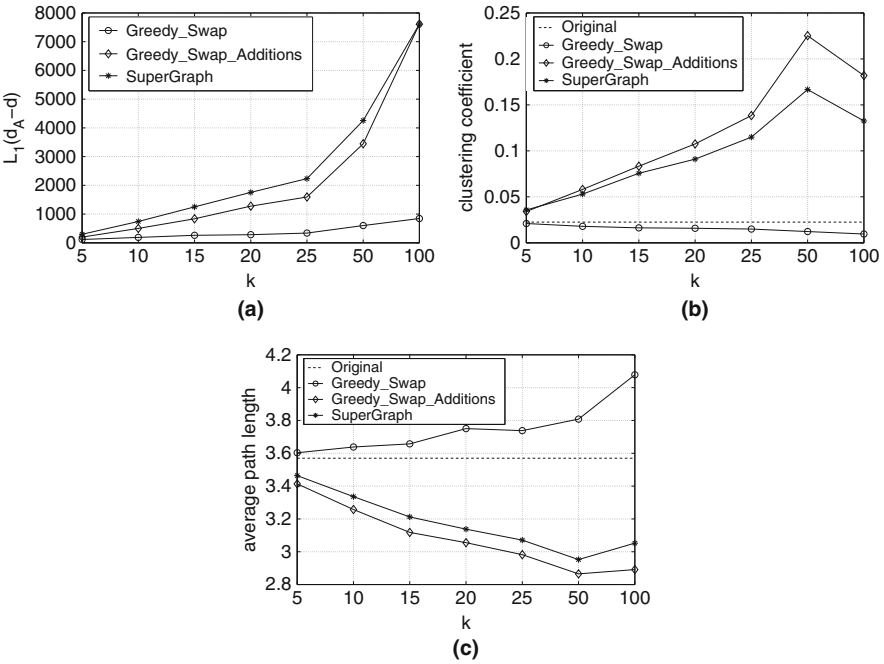


Fig. 14.4 Synthetic data sets: scale-free graphs $G_{BS} \in \mathcal{G}_{BS}$. (**a**) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k (**b**) CC as a function of k (**c**) APL as a function of k

plots, there is a constant line appearing; this corresponds to the value of the clustering coefficient of the *original* graph, which is unaffected by the value of k. Note that all the plots show that the values of the clustering coefficient, though different in the degree anonymous graphs, never deviate too much from their original values; the largest difference in the CC values from the original values is observed for the *co-author* data set, where the difference is 0.24 for the degree anonymous graph produced by the `Greedy_Swap_Additions` algorithm when $k = 100$. But even in this case, the other two algorithms output graphs with CC almost equal to that of the original graph. Note that there is no clear trend on how the CC changes when the graph becomes degree anonymous. Both increments and decrements are observed; however the changes are generally negligible.

Average path length (APL)
In Figs. 14.3–14.6 we report the values of the average path length of the degree anonymous graphs and the original graphs. As expected, the anonymization process of `Greedy_Swap_Additions` and `Supergraph` decreases the average path length of the output graph since only new connections are added. The `Greedy_Swap`, on the other hand, can either increase or decrease the average path length because it simultaneously adds and deletes the edges.

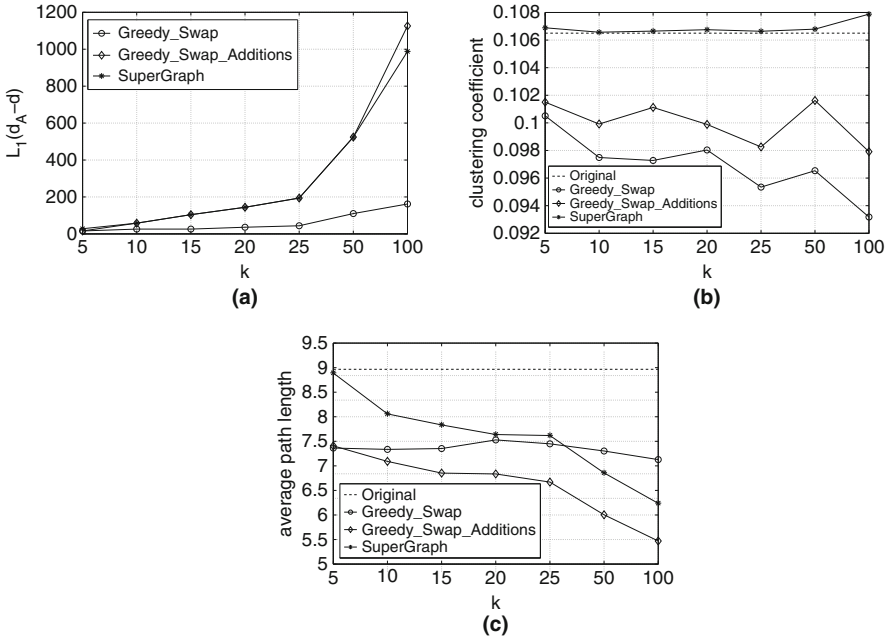


Fig. 14.5 Real data sets datasets: *powergrid* data. (**a**) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k (**b**) CC as a function of k (**c**) APL as a function of k

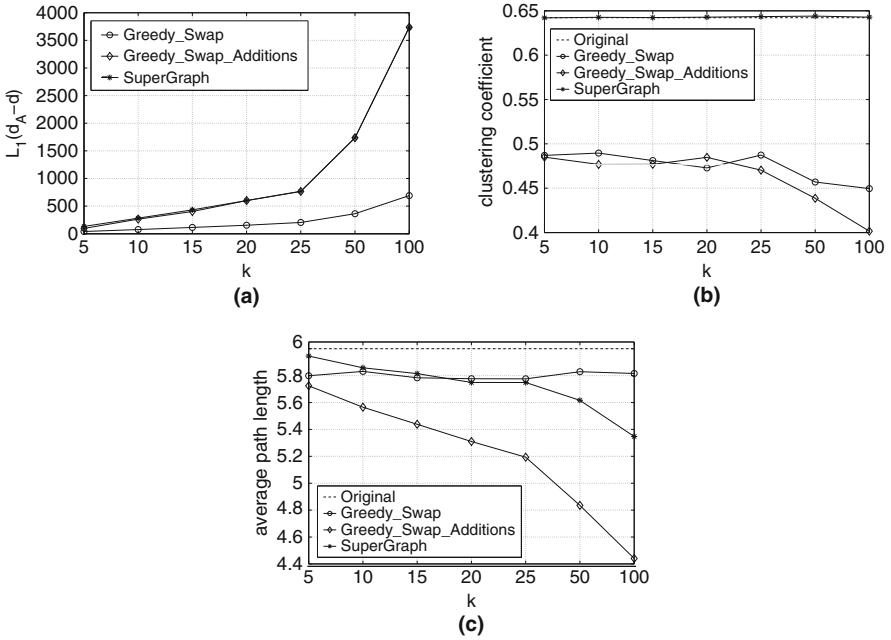


Fig. 14.6 Real data sets datasets: *co-authors* data. (**a**) $L_1(\mathbf{d}_A - \mathbf{d})$ as a function of k (**b**) CC as a function of k (**c**) APL as a function of k

Very similar results have been obtained for other data sets generated using the random-graph model as well as the *enron* data set. However, we omit the corresponding plots due to space constraints.

Structural difference

Recall that the structural difference between two graphs $G(V, E)$ and $\widehat{G}(V, \widehat{E})$, $\Delta\left(\widehat{G}, G\right)$, is the symmetric difference of their edge sets, i.e., $\Delta\left(\widehat{G}, G\right) := \left|\widehat{E} \setminus E\right| + \left|E \setminus \widehat{E}\right|$. Table 14.2 summarizes the values of $\Delta\left(\widehat{G}, G\right)$ obtained by different algorithms for the *co-authors* data. It is interesting to observe that `Supergraph` consistently has a low value. This is due to the reason that `Supergraph` constructs the anonymized graph by only extending the input graph, and therefore, $\left|E \setminus \widehat{E}\right|$ is always 0. On the other hand, both `Greedy_Swap` and `Greedy_Swap_Additions` construct an anonymized graph from scratch and then heuristically add and delete edges to make it more similar to the original graph. This procedure leads to higher symmetric differences of their edge sets.

Table 14.2 Structural differences between G and $\widehat{G}$, obtained by different algorithms for the *co-authors* data

	`Greedy_Swap`	`Greedy_Swap_Additions`	`Supergraph`
$k = 5$	1714	1764	66
$k = 10$	1742	1904	141
$k = 15$	1846	2009	216
$k = 20$	1815	2126	300
$k = 25$	1868	2269	384
$k = 50$	2096	3068	868
$k = 100$	2232	4402	1868

However, readers should not be deluded by the straight numbers in the table and reach a conclusion that `Greedy_Swap` and `Greedy_Swap_Additions` are significantly inferior to `Supergraph`. In fact, we can show that the number of edges that are added and deleted by these two greedy algorithms only accounts for a very small portion of the overall edge sets. To illustrate this, we decompose the structural difference between G and $\widehat{G}$ into two normalized components: $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$ and $\frac{|E \setminus \widehat{E}|}{|E|}$. The former gives the percentage of the edges in the anonymized graph that are newly added, and the latter calculates the percentage of the edges in the original graph that are deleted. The lower these two values, the better the structure of the original graph is preserved. Figure 14.7a and 14.7b shows the values of $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$ and $\frac{|E \setminus \widehat{E}|}{|E|}$ obtained by different algorithm as a function of $k = \{5, 10, 15, 20, 25, 50, 100\}$ for the *co-authors* data. We can easily observe that both `Greedy_Swap` and `Greedy_Swap_Additions` produce very small values of $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$ and $\frac{|E \setminus \widehat{E}|}{|E|}$. In particular, `Greedy_Swap` achieves 0.12 for both components even when k is 100, which is better than `Supergraph` that has a value of 0.16 for $\frac{|\widehat{E} \setminus E|}{|\widehat{E}|}$.

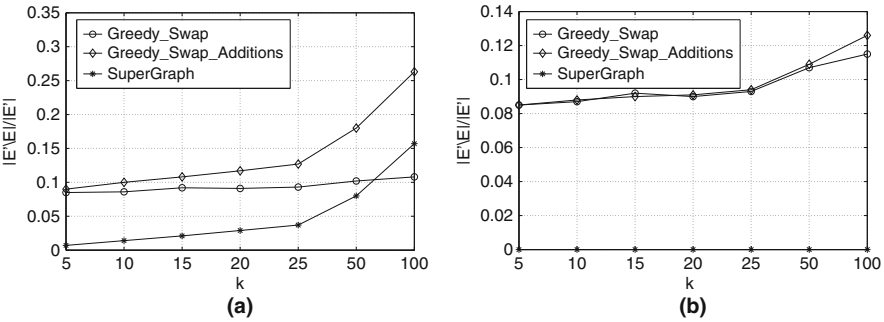


Fig. 14.7 Decomposed structural differences between G and $\widehat{G}$, obtained by different algorithms for the *co-authors* data. (**a**) $\frac{|\widehat{E}\setminus E|}{|\widehat{E}|}$ as a function of k (**b**) $\frac{|E\setminus\widehat{E}|}{|E|}$ as a function of k

14.7.2.1 Exploring the Scale-Free Graphs

Previous work on the analysis of complex networks has shown that many of the real-world graphs are scale free, i.e., their node degrees follow a power-law distribution. In this section we demonstrate that our anonymization framework does not destroy the power-law property of the original graph if k is not too big. That is, if the input graph has a power-law degree distribution, so does the degree anonymous version of it.

In Table 14.3, we report the values of the estimated exponent (γ) of the power-law distribution of the original *co-authors* data and its degree anonymous counterpart. We can observe that the new γ values obtained by all three algorithms exhibit high degree of similarity to the original one for $k < 15$. When k gets larger, `Greedy_Swap` still preserves the γ value very well. This result is due to the fact that the degree-sequence anonymization of `Greedy_Swap` minimally changes the degree sequence of a graph. For significant large values of k (e.g., $k = 100$), a great amount of the nodes in the anonymized graph will have the same degree, and the power-law distribution will change. We claim that this is a natural result for any degree anonymization algorithm.

Table 14.3 Real data set: *co-authors* graph. Value of the exponent (γ) of the power-law distribution of the original and the k-degree anonymous graph obtained using `Greedy_Swap`, `Greedy_Swap_Additions` and `Supergraph` algorithms, for $k = 10, 15, 20, 25, 50, 100$

	γ		
	`Greedy_Swap`	`Greedy_Swap_Additions`	`Supergraph`
Original	**2.07**	**2.07**	**2.07**
$k = 10$	2.45	2.26	2.26
$k = 15$	2.33	2.13	2.13
$k = 20$	2.28	1.97	1.97
$k = 25$	2.25	1.83	1.83
$k = 50$	2.05	1.57	1.57
$k = 100$	1.92	1.22	1.22

14.8 Conclusions

The degree of a node in a graph, among other structural characteristics, can to a large extent distinguish the node from other nodes. In this chapter, we focused on a specific graph-anonymity notion that prevents the re-identification of individuals by an attacker with certain prior knowledge of the degrees. We formally defined the GRAPH ANONYMIZATION problem that, given an input graph, asks for the graph modifications (in terms of additions and deletions of edges) that allow for the transformation of the input to a degree anonymous graph, i.e., a graph in which every node shares the same degree with $k-1$ other nodes. We showed that this problem can be decomposed into two subproblems and proposed simple and efficient algorithms for solving them. We also presented experiments on synthetic and real-world graph data and demonstrated the utility of the degree anonymous graphs as well as the efficiency of our methods.

From the material presented in this chapter as well as in the related work (presented in Section 3) it becomes apparent that privacy-preserving data analysis on graph data raises many more challenges when compared to the challenges that arise from anonymization and perturbation techniques in tabular data. In the latter case, each tuple can be viewed as an independent sample from some distribution. However, in a graph, all the nodes and edges are correlated; a single change of an edge and/or a node can spread across the whole network. Moreover, in graphs it is difficult to model the capability of an attacker. In principle, any topological structure of the graph can be potentially used to derive private information. Another challenge is related to the right definition of the utility of an anonymized graph. For example, we measured the utility using the degree anonymization and the structural cost. However, other measures that associate the structural properties of the original and the anonymized graph can also be used. Further, these measures can be also tailored to a particular graph-analysis task. Overall, there are many directions that need to be explored before the research community agrees upon a unifying, theoretically and practically sound model for privacy in social networks.

References

1. C. C. Aggarwal, and P. S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*, vol. 34 of *Advances in Database Systems*. Springer, 233 Spring Street, New York, NY 10013, USA, 2008.
2. L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou R3579X?: Anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)* (Alberta, Canada, May 2007), pp. 181–190.
3. A.-L. Barabási, and R. Albert. Emergence of scaling in random networks. *Science 286*, 5439 (October 1999), 509–512.
4. R. J. Bayardo, and R. Agrawal. Data privacy through optimal k-anonymization. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 217–228 Tokyo, Japan, April 2005.

5. S. Boyd, and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2004.
6. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.
7. I. Diakonikolas, and M. Yannakakis. Succinct approximate convex pareto curves. In *SODA*, pages 74–83 2008.
8. P. Erdös, and T. Gallai. Graphs with prescribed degrees of vertices. *Mat. Lapok*, 11:264–274, 1960.
9. L. Getoor, and C. P. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter,* 7, (2): 3–12, 2005.
10. S. L. Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. *Journal of the Society for Industrial and Applied Mathematics*, 10(3): 496–506, 1962.
11. M. Hay, G. Miklau, D. Jensen, D. Towsely, and P. Weis. Resisting structural re-identification in anonymized social networks. In *Proceedings of the VLDB Endowment*. Volume 1, Issue 1, pages 102–114, Publisher VLDB Endowment, August 2008.
12. M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava. Anonymizing social networks. Technical report, University of Massachusetts Amherst, 2007.
13. A. Korolova, R. Motwani, S. U. Nabar, and 0002, Y. X. Link privacy in social networks. In *CIKM*, pages 289–298, 2008.
14. Y.-S. Lee. Graphical demonstration of an optimality property of the median. *The American Statistician* 49(4): 369–372, November 1995.
15. K. Liu, and E. Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, pages 93–106, Vancouver, Canada, June 2008.
16. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, pages 24, Atlanta, GA, April 2006, 2006.
17. A. Meyerson, and R. Williams. On the complexity of optimal k-anonymity. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'04)*, pages 223–228, Paris, France, 2004, 2004.
18. J. Pei, and B. Zhou. Preserving privacy in social networks against neighborhood attacks. In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08)*, Cancun, Mexico, April 2008.
19. P. Samarati, and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)* page 188 Seattle, WA, 1998.
20. D. J. Watts. Networks, dynamics, and the small-world phenomenon. *American Journal of Sociology*, 105(2): 493–527, September 1999.
21. D. J. Watts, and S. H. Strogatz. Collective dynamics of small-world networks. *Nature* 393 (6684): 409–410, June 1998.
22. X. Ying, and X. Wu. Randomizing social networks: a spectrum preserving approach. In *Proceedings of SIAM International Conference on Data Mining (SDM'08)*, pages 739–750, Atlanta, GA, April 2008.
23. E. Zheleva, and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *Proceedings of the International Workshop on Privacy, Security, and Trust in KDD (PinKDD'07)*, pages 153–171, San Jose, CA, August 2007.
24. B. Zhou, J. Pei, and W.-S. Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *ACM SIGKDD Explorations* 10(2): 12–22, December, 2008.

Part V
Summarization and OLAP of Information Networks

Chapter 15
Interactive Graph Summarization

Yuanyuan Tian and Jignesh M. Patel

Abstract Graphs are widely used to model real-world objects and their relationships, and large graph data sets are common in many application domains. To understand the underlying characteristics of large graphs, graph summarization techniques are critical. Existing graph summarization methods are mostly statistical (studying statistics such as degree distributions, hop-plots, and clustering coefficients). These statistical methods are very useful, but the resolutions of the summaries are hard to control. In this chapter, we introduce database-style operations to summarize graphs. Like the OLAP-style aggregation methods that allow users to interactively drill-down or roll-up to control the resolution of summarization, the methods described in this chapter provide an analogous functionality for large graph data sets.

15.1 Introduction

Graphs provide a powerful primitive for modeling real-world objects and the relationships between objects. Various modern applications have generated large amount of graph data. Some of these application domains are listed below:

- Popular social networking web sites, such as Facebook (www.facebook.com), MySpace (www.myspace.com), and LinkedIn (www.linkedin.com), attract millions of users (nodes) connected by their friendships (edges). By April 2009, the number of active users on Facebook has grown to 200 million, and on average each user has 120 friends. Mining these social networks can provide valuable information on social relationships and user communities with common interests. Besides mining the friendship network, one can also mine the "implicit" interaction network formed by dynamic interactions (such as sending a message to a friend).

J.M. Patel (✉)
University of Wisconsin Madison, WI, USA
e-mail: jignesh@cs.wisc.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_15,

- Coauthorship networks and citation networks constructed from DBLP (www.informatik.uni-trier.de/~ley/db/) and CiteSeer (citeseer.ist.psu.edu) can help understand publication patterns of researchers.
- Market basket data, such as those produced from Amazon (www.amazon.com) and Netflix (www.netflix.com), contain information about millions of products purchased by millions of customers, which forms a bipartite graph with edges connecting customers to products. Exploiting the graph structure of the market basket data can improve customer segmentation and targeted advertising.
- The link structure of the World Wide Web can be naturally represented as a graph with nodes representing web pages and directed edges representing the hyperlinks. According to the estimate at www.worldwidewebsize.com, by May 15, 2009, the World Wide Web contains at least 30.05 billion webpages. The graph structure of the World Wide Web has been extensively exploited to improve search quality [8], discover web communities [16], and detect link spam[23].

With the overwhelming wealth of information encoded in these graphs, there is a critical need for tools to summarize large graph data sets into concise forms that can be easily understood.

Graph summarization has attracted a lot of interest from a variety of research communities, including sociology, physics, and computer science. It is a very broad research area that covers many topics. Different ways of summarizing and understanding graphs have been invented across these different research communities. These different summarization approaches extract graph characteristics from different perspectives and are often complementary to each other. Sociologists and physicists mostly apply statistical methods to study graph characteristics. The summaries of graphs are statistical measures, such as degree distributions for investigating the scale-free property of graphs, hop-plots for studying the small world effect, and clustering coefficients for measuring the clumpiness of large graphs. Some examples of this approach were presented in Chapter 8. In the database research community, methods for mining frequent subgraph patterns are used to understand the characteristics of large graphs, which was the focus of Chapter 4. The summaries produced by these methods are sets of frequently occurring subgraphs (in the original graphs). Various graph clustering (or partitioning) algorithms are used to detect community structures (dense subgraphs) in large graphs. For these methods, the summaries that are produced are partitions of the original graphs. This topic is covered in Chapters 3 and 7. Graph compression and graph visualization are also related to the graph summarization problem. These two topics will be discussed in Section 15.7 of this chapter.

This chapter, however, focuses on a graph summarization method that produces small and informative summaries, which themselves are also graphs. We call them *summary graphs*. These summary graphs are much more compact in size and provide valuable insight into the characteristics of the original graphs. For example, in Fig. 15.1, a graph with 7445 nodes and 19,971 edges is shown on the left. Understanding this fairly small graph by mere visual inspection of the raw graph structure is very challenging. However, the summarization method introduced in this chapter

will generate much compact and informative graphs that summarize the high-level structure characteristics of the original graph and the dominant relationships among clusters of nodes. An example summary graph for the original graph is shown on the right of Fig. 15.1. In the summary graph, each node represents a set of nodes from the original graph, and each edge of the summary graph represents the connections between two corresponding sets of nodes. The formal definition of summary graphs will be introduced in Section 15.2.

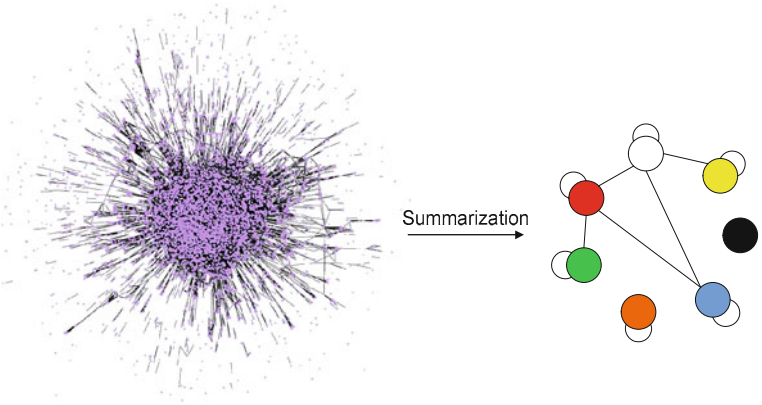


Fig. 15.1 A summary graph (*right*) is generated for the original graph (*left*)

The concept of summary graph is the foundation of the summarization method presented in this chapter. This method is very unique in that it is amenable to an interactive querying scheme by allowing users to customize the summaries based on user-selected node attributes and relationships. Furthermore, this method empowers users to control the resolutions of the resulting summaries, in conjunction with an intuitive "drill-down" or "roll-up" paradigm to navigate through summaries with different resolution. This last aspect of drill-down or roll-up capability is inspired by the OLAP-style aggregation methods [11] in the traditional database systems.

Note that the method introduced in this chapter is applicable for both directed and undirected graphs. For ease of presentation, we only consider undirected graphs in this chapter.

The remainder of this chapter is organized as follows: We first introduce the formal definition of summary graph in Section 15.2, then discuss the aggregation-based graph summarization method in Section 15.3. Section 15.4 shows an interesting example of applying this graph summarization method to the DBLP [17] coauthorship graph. Section 15.5 demonstrates the scalability of the described method. Section 15.6 provides some discussion on the summarization method. Related topics, such as graph compression and graph visualization are discussed in Section 15.7. Finally, Section 15.8 concludes this chapter.

15.2 Summary Graphs

In this chapter, we consider a general graph model where nodes in the graph have arbitrary number of associated attributes and are connected by multiple types of edges. More formally, a graph is denoted as $G = (V, E)$, where V is the set of nodes and E is the set of edges. The set of attributes associated with the nodes is denoted as $A = \{a_1, a_2, \ldots, a_m\}$. We require that each node $v \in V$ has a value for every attribute in A. The set of edge types present in the graph is denoted as $T = \{t_1, t_2, \ldots, t_n\}$. Each edge $(u, v) \in E$ can be marked by a non-trivial subset of edge types denoted as $T(u, v)$ ($\emptyset \subset T(u, v) \subseteq T$). For example, Fig. 15.2a shows a sample social networking graph. In this graph, nodes represent students. Each student node has attributes such as gender and department. In addition, there are two types of relationships present in this graph: friends and classmates. While some students are only friends or classmates with each other, others are connected by both relationships. Note that in this figure, only a few edges are shown for compactness.

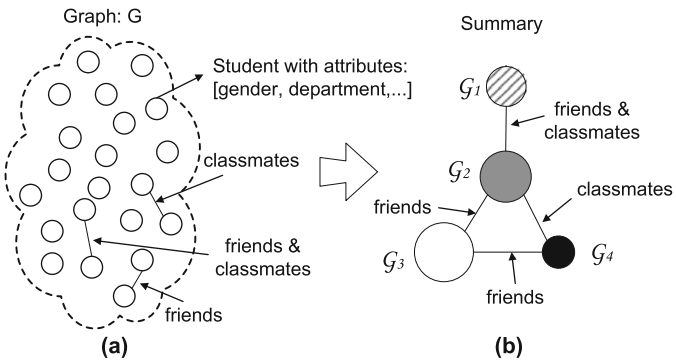


Fig. 15.2 Graph summarization by aggregation

We can formally define summary graphs as follows: Given a graph $G = (V, E)$, and a partition of V, $\Phi = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k\}$ ($\bigcup_{i=1}^{k} \mathcal{G}_i = V$ and $\forall i \neq j\ \mathcal{G}_i \cap \mathcal{G}_j = \emptyset$), the *summary graph* based on Φ is $S = (V_S, E_S)$, where $V_S = \Phi$, and $E_S = \{(\mathcal{G}_i, \mathcal{G}_j) | \exists u \in \mathcal{G}_i, v \in \mathcal{G}_j, (u, v) \in E\}$. The set of edge types for each $(\mathcal{G}_i, \mathcal{G}_j) \in E_S$ is defined as $T(\mathcal{G}_i, \mathcal{G}_j) = \bigcup_{(u,v)\in E, u\in\mathcal{G}_i, v\in\mathcal{G}_j} T(u, v)$.

More intuitively, each node of the summary graph, called a *group* or a *supernode*, corresponds to one group in the partition of the original node set, and an edge, called *group relationships* or *superedges*, represents the connections between two corresponding sets of nodes. A group relationship between two groups exists if and only if there exists at least one edge connecting some nodes in the two groups. The set of edge types for a group relationship is the union of all the types of the corresponding edges connecting nodes in the two groups.

15.3 Aggregation-Based Graph Summarization

The graph summarization method we will discuss in this chapter is a database-style graph aggregation approach [28]. This aggregation-based graph summarization approach contains two operations.

The first operation, called *SNAP* (*S*ummarization by Grouping *N*odes on *A*ttributes and *P*airwise Relationships), produces a summary graph of the input graph by grouping nodes based on user-selected node attributes and relationships. The *SNAP* summary for the graph in Fig. 15.2a is shown in Fig. 15.2b. This summary contains four groups of students and the relationships between these groups. Students in each group have the same gender and are in the same department, and they relate to students belonging to the same set of groups with friends and classmates relationships. This compact summary reveals the underlying characteristics about the nodes and their relationships in the original graph.

The second operation, called *k-SNAP*, further allows users to control the resolutions of summaries. This operation is pictorially depicted in Fig. 15.3. Here using the slider, a user can "drill-down" to a larger summary with more details or "roll-up" to a smaller summary with less details.

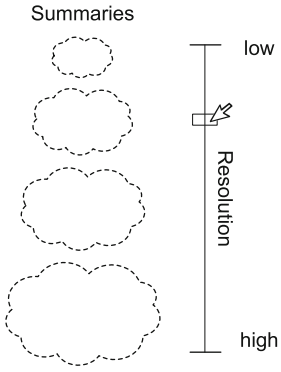


Fig. 15.3 Illustration of multi-resolution summaries

Next we describe the two operations in more detail and present algorithms to evaluate the *SNAP* and *k-SNAP* operations.

15.3.1 SNAP Operation

The *SNAP* operation produces a summary graph through a homogeneous grouping of the input graph's nodes, based on user-selected node attributes and relationships. Figure 15.2b shows an example summary graph for the original graph in Fig. 15.2a, generated by the *SNAP* operation based on gender and department attributes, and classmates and friends relationships.

The summary graph produced by *SNAP* operation has to satisfy the following three requirements:

Attributes Homogeneity: Nodes in each group have the same value for each user-selected attribute.

Relationships Homogeneity: Nodes in each group connect to nodes belonging to the same set of groups with respect to each type of user-selected relationships. For example, in Fig. 15.2b, every student (node) in group $\mathcal{G}_2$ is a friend of some student(s) in $\mathcal{G}_3$, a classmate of some student(s) in $\mathcal{G}_4$, and has at least a friend as well as a classmate in $\mathcal{G}_1$. By the definition of relationships homogeneity, for each pair of groups in the result of the *SNAP* operation, if there is a group relationship between the two, then every node in both groups has to participate in this group relationship (i.e. every node in one group connects to at least one node in the other group).

Minimality: The number of groups in the summary graph is the minimal among all possible groupings that satisfy attributes homogeneity and relationships homogeneity requirements.

There could be more than one grouping satisfying the attributes homogeneity and relationships homogeneity requirements. In fact, the grouping in which each node forms a group is always compatible with any given attributes and relationships (see [28] for more details). The minimality requirement guarantees that the summary graph is the most compact in size.

15.3.1.1 Evaluating *SNAP* Operation

The *SNAP* summary graph can be produced by the following top-down approach:

Top-down *SNAP* Approach

Step 1: Partition nodes based only on the user-selected attributes.
Step 2: Iteration Step
while a group breaks the relationships homogeneity requirement **do**
Split the group based on its relationships with other groups
end while

In the first step of the top-down *SNAP* approach, nodes in the original graph are partitioned based only on the user-selected attributes. This step guarantees that further splits of the grouping always satisfy the attributes homogeneity requirement. For example, if nodes in a graph only have one attribute with values A, B, and C, then the first step produces a partition of three groups with attribute values A, B, and C, respectively, such as the example grouping shown in Fig. 15.4a. In the iterative step, the algorithm checks whether the current grouping satisfies the relationships homogeneity requirement. If not, the algorithm picks a group that breaks the requirement and splits this group based on how the nodes in this group connect to nodes in other groups. In the example shown in Fig. 15.4a, group A (the group

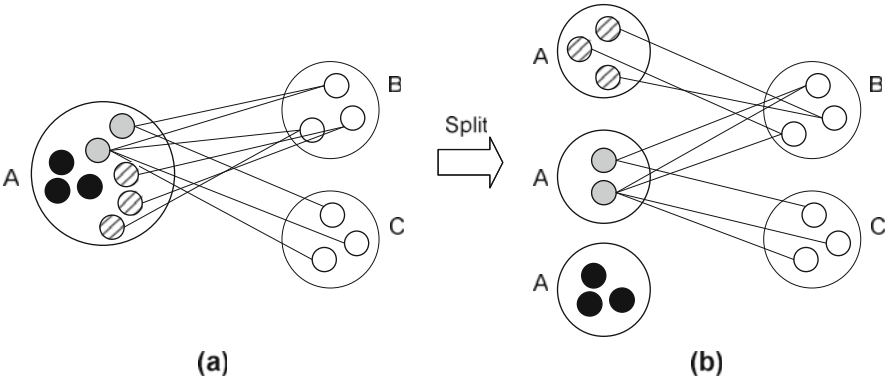


Fig. 15.4 An example of splitting a group based on its relationships with other groups in the top-down *SNAP* approach

with attribute value A) does not satisfy the relationships homogeneity requirement: the black nodes do not connect to any nodes in other groups, the shaded nodes only connect to nodes in group B, while the gray nodes connect to both group B and group C. This group is split based on how the nodes in this group connect to nodes in other groups, which ends up with three subgroups containing black nodes, gray nodes, and shaded nodes shown in Fig. 15.4b. In the next iteration, the algorithm continues to check whether the new grouping satisfies the homogeneity requirement. If not, it selects and splits a group that breaks the requirement. The iterative process continues until the current grouping satisfies the relationships homogeneity requirement. It is easy to prove that the grouping resulting from this algorithm contains the minimum number of groups satisfying both the attributes and relationships homogeneity requirements.

15.3.2 k-SNAP Operation

15.3.2.1 Limitations of the *SNAP* Operation

The *SNAP* operation produces a grouping in which nodes of each group are homogeneous with respect to user-selected attributes and relationships. Unfortunately, homogeneity is often too restrictive in practice, as most real life graph data are subject to noise and uncertainty; for example, some edges may be missing, and some edges may be spurious because of errors. Applying the *SNAP* operation on noisy data can result in a large number of small groups, and, in the worst case, each node may end up in an individual group. Such a large summary is not very useful in practice. A better alternative is to let users control the sizes of the results to get summaries with the resolutions that they can manage (as shown in Fig. 15.3).

The *k-SNAP* operation is introduced to relax the homogeneity requirement for the relationships and allow users to control the sizes of the summaries.

The relaxation of the homogeneity requirement for the relationships is based on the following observation. For each pair of groups in the result of the *SNAP* operation, if there is a group relationship between the two, then every node in both groups participates in this group relationship. In other words, every node in one group relates to some node(s) in the other group. On the other hand, if there is no group relationship between two groups, then absolutely no relationship connects any nodes across the two groups. However, in reality, if most (not all) nodes in the two groups participate in the group relationship, it is often a good indication of a "strong" relationship between the two groups. Likewise, it is intuitive to mark two groups as being "weakly" related if only a small fraction of nodes are connected between these groups.

Based on these observations, the *k-SNAP* operation relaxes the homogeneity requirement for the relationships by not requiring that every node participates in a group relationship. But it still maintains the homogeneity requirement for the attributes, i.e., all the groupings should be homogeneous with respect to the given attributes. Users control how many groups are present in the summary by specifying the required number of groups, denoted as k. There are many different groupings of size k, thus there is a need to measure the qualities of the different groupings. The Δ-measure is proposed to assess the quality of a *k-SNAP* summary by examining how different it is to a hypothetical "ideal summary."

15.3.2.2 Measuring the Quality of *k-SNAP* Summaries

We first define the set of nodes in group $\mathcal{G}_i$ that participate in a group relationship $(\mathcal{G}_i, \mathcal{G}_j)$ as $P_{\mathcal{G}_j}(\mathcal{G}_i) = \{u | u \in \mathcal{G}_i \text{ and } \exists v \in \mathcal{G}_j \text{ s.t. } (u, v) \in E\}$. Then we define the participation ratio of the group relationship $(\mathcal{G}_i, \mathcal{G}_j)$ as $p_{i,j} = \frac{|P_{\mathcal{G}_j}(\mathcal{G}_i)|+|P_{\mathcal{G}_i}(\mathcal{G}_j)|}{|\mathcal{G}_i|+|\mathcal{G}_j|}$. For a group relationship, if its participation ratio is greater than 50%, we call it a strong group relationship, otherwise, we call it a weak group relationship. Note that in a *SNAP* summary, the participation ratios are either 100 or 0%.

Given a graph G, the Δ-measure of a grouping of nodes $\Phi = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k\}$ is defined as follows:

$$\Delta(\Phi) = \sum_{\mathcal{G}_i, \mathcal{G}_j \in \Phi} (\delta_{\mathcal{G}_j}(\mathcal{G}_i) + \delta_{\mathcal{G}_i}(\mathcal{G}_j)), \tag{15.1}$$

where

$$\delta_{\mathcal{G}_j}(\mathcal{G}_i) = \begin{cases} |P_{\mathcal{G}_j}(\mathcal{G}_i)| & \text{if } p_{i,j} \leq 0.5 \\ |\mathcal{G}_i| - |P_{\mathcal{G}_j}(\mathcal{G}_i)| & \text{otherwise.} \end{cases} \tag{15.2}$$

Note that if the graph contains multiple types of relationships, then the Δ value for each edge type is aggregated as the final Δ value.

Intuitively, the Δ-measure counts the minimum number of differences in participations of group relationships between the given *k-SNAP* grouping and a

hypothetical ideal grouping of the same size in which all the strong relationships have 100% participation ratio and all the weak relationships have 0% participation ratio. The measure looks at each pairwise group relationship: If this group relationship is weak ($p_{i,k} \leq 0.5$), then it counts the participation differences between this weak relationship and a non-relationship ($p_{i,k} = 0$); on the other hand, if the group relationship is strong, it counts the differences between this strong relationship and a 100% participation-ratio group relationship. The δ function, defined in (15.2), evaluates the part of the Δ value contributed by a group $\mathcal{G}_i$ with one of its neighbors $\mathcal{G}_j$ in a group relationship. Note that $\delta_{\mathcal{G}_i}(\mathcal{G}_i)$ measures the contribution to the Δ value by the connections within the group $\mathcal{G}_i$ itself.

Given this quality measure and the user-specified resolution k (i.e. number of groups in the summary is k), the goal of the *k-SNAP* operation is to find the summary of size k with the best quality. However, this problem has been proved to be NP-Complete [28]. Two heuristic-based algorithms are proposed to evaluate the *k-SNAP* operation approximately.

Top-Down *k-SNAP* Approach

Similar to the top-down *SNAP* algorithm, the top-down *k-SNAP* approach also starts from the grouping based only on attributes, and then iteratively splits existing groups until the number of groups reaches k.

However, in contrast to the *SNAP* evaluation algorithm, which randomly chooses a splittable group and splits it into subgroups based on its relationships with other groups, the top-down approach has to make the following decisions at each iterative step: (1) which group to split and (2) how to split it. Such decisions are critical as once a group is split, the next step will operate on the new grouping. At each step, the algorithm can only make the decision based on the current grouping. Each step should make the smallest move possible, to avoid going too far away from the right direction. Therefore, the algorithm splits one group into only two subgroups at each iterative step. There are different ways to split one group into two. One natural way is to divide the group based on whether nodes have relationships with nodes in a neighbor group. After the split, nodes in the two new groups either all or never participate in the group relationships with this neighbor group.

As discussed in Section 15.3.2.2, the *k-SNAP* operation tries to find the grouping with a minimum Δ measure (see (15.1)) for a given k. The computation of the Δ measure can be broken down into each group with each of its neighbors (see the δ function in (15.2)). Therefore, our heuristic chooses the group that makes the most contribution to the Δ value with one of its neighbor groups. More formally, for each group $\mathcal{G}_i$, we define $CT(\mathcal{G}_i)$ as follows:

$$CT(\mathcal{G}_i) = \max_{\mathcal{G}_j}\{\delta_{\mathcal{G}_j}(\mathcal{G}_i)\}. \tag{15.3}$$

Then, at each iterative step, we always choose the group with the maximum CT value to split and then split it based on whether nodes in this group $\mathcal{G}_i$ have relationships with nodes in its neighbor group $\mathcal{G}_t$, where

$$\mathcal{G}_t = arg \max_{\mathcal{G}_j}\{\delta_{\mathcal{G}_j}(\mathcal{G}_i)\}.$$

The top-down *k-SNAP* approach can be summarized as follows:

Top-down *k-SNAP* Approach

Step 1: Partition nodes based only on the user selected attributes.
Step 2: Iteration Step
while the grouping size is less than k **do**
Find $\mathcal{G}_i$ with the maximum $CT(\mathcal{G}_i)$ value
Split $\mathcal{G}_i$ based on its relationship with $\mathcal{G}_t = arg \max_{\mathcal{G}_j}\{\delta_{\mathcal{G}_j}(\mathcal{G}_i)\}$
end while

Bottom-Up *k-SNAP* Approach

The bottom-up approach first computes the *SNAP* summary using the top-down *SNAP* approach. The *SNAP* summary is the summary with the finest resolution, as the participation ratios of group relationships are either 100 or 0%. Starting from the finest summary, the bottom-up approach iteratively merges two groups until the number of groups reduces to k.

Choosing which two groups to merge in each iterative step is crucial for the bottom-up approach. First, the two groups are required to have the same attribute values. Second, the two groups must have similar group relationships with other groups. Now, this similarity between two groups can be formally defined as follows.

The two groups to be merged should have similar neighbor groups with similar participation ratios. We define a measure called $MergeDist$ to assess the similarity between two groups in the merging process.

$$MergeDist(\mathcal{G}_i, \mathcal{G}_j) = \sum_{k \neq i,j} |p_{i,k} - p_{j,k}|. \tag{15.4}$$

$MergeDist$ accumulates the differences in participation ratios between $\mathcal{G}_i$ and $\mathcal{G}_j$ with other groups. The smaller this value is, the more similar the two groups are.

The bottom-up *k-SNAP* approach can be summarized as follows:

Bottom-up *k-SNAP* Approach

Step 1: Compute the *SNAP* summary using the top-down *SNAP* approach
Step 2: Iteration Step
while the grouping size is greater than k **do**
Find $\mathcal{G}_i$ and $\mathcal{G}_j$ with the minimum $MergeDist(\mathcal{G}_i, \mathcal{G}_j)$ value
Merge $\mathcal{G}_i$ and $\mathcal{G}_j$
end while

15.3.3 Top-Down k-SNAP Approach vs. Bottom-Up k-SNAP Approach

In [28], extensive experiments show that the top-down *k-SNAP* approach significantly outperforms the bottom-up *k-SNAP* approach in both effectiveness and efficiency for small k values. In practice, users are more likely to choose small k values to generate summaries. Therefore, the top-down approach is preferred for most practical uses.

15.4 An Example Application on Coauthorship Graphs

This section presents an example of applying the aggregation-based graph summarization approach to analyze the coauthorship graph of database researchers. The database researcher coauthorship graph is generated from the DBLP Bibliography data [17] by collecting the publications of a number of selected journals and conferences in the database area. The constructed coauthorship graph with 7445 authors and 19,971 coauthorships is shown in Fig. 15.5. Each node in this graph represents an author and has an attribute called *PubNum*, which is the number of publications belonging to the corresponding author. Another attribute called *Prolific* is assigned to each author in the graph indicating whether that author is prolific: authors with ≤ 5 papers are tagged as low prolific (LP), authors with > 5 but ≤ 20 papers are prolific (P), and the authors with > 20 papers are tagged as highly prolific (HP).

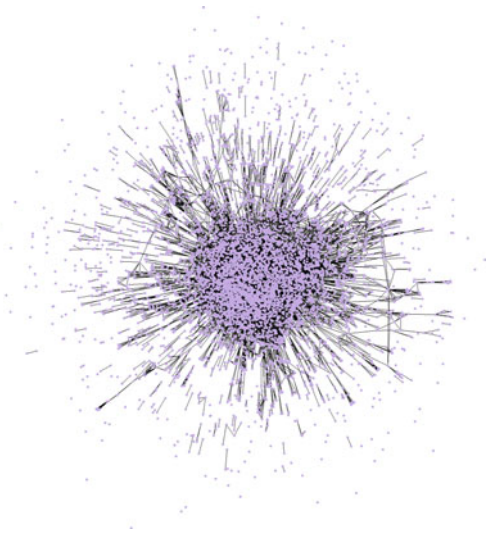


Fig. 15.5 DBLP DB coauthorship graph

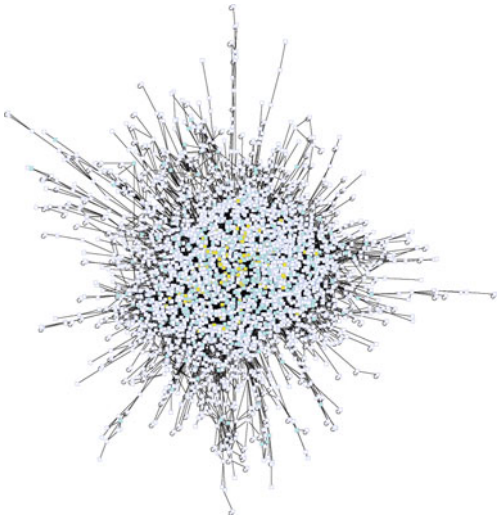


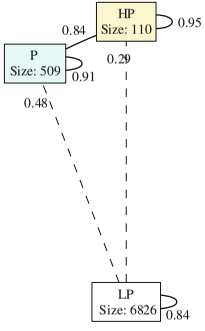
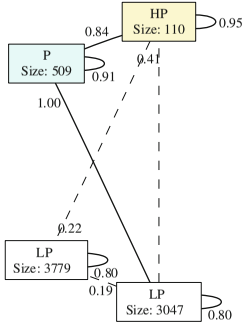
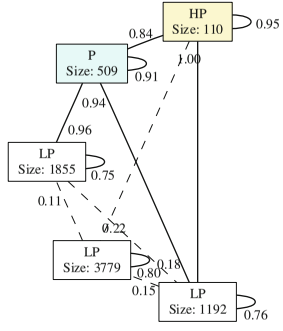
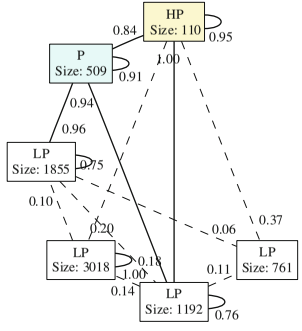
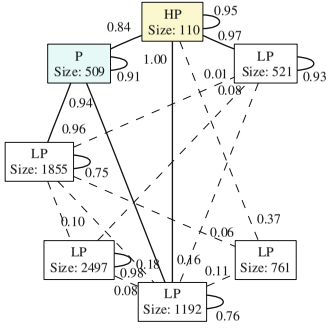
Fig. 15.6 The *SNAP* result for DBLP DB coauthorship graph

The *SNAP* operation on the *Prolific* attribute and the coauthorships produces a summary graph shown in Fig. 15.6. The *SNAP* operation results in a summary with 3569 groups and 11,293 group relationships. This summary is too big to analyze. On the other hand, by applying the *SNAP* operation on only the *Prolific* attribute (i.e., not considering any relationships in the *SNAP* operation), a summary with only three groups is produced shown in the leftmost figure of Table 15.1. The bold edges between two groups indicate strong group relationships (with more than 50% participation ratio), while dashed edges are weak group relationships. This summary shows that the HP researchers as a whole have very strong coauthorship with the P group of researchers. Researchers within both groups also tend to coauthor with people within their own groups. However, this summary does not provide a lot of information for the LP researchers: they tend to coauthor strongly within their group and they have some connection with the HP and P groups.

Now, making use of the *k-SNAP* operation, summaries with multiple resolutions are generated. The figures in Table 15.1 show the *k-SNAP* summaries for $k = 4, 5, 6,$ and 7. As k increases, more details are shown in the summaries.

When $k = 7$, the summary shows that there are five subgroups of LP researchers. One group of 1192 LP researchers strongly collaborates with both HP and P researchers. One group of 521 only strongly collaborates with HP researchers. One group of 1855 only strongly collaborates with P researchers. These three groups also strongly collaborate within their groups. There is another group of 2497 LP researchers that has very weak connections to other groups but strongly cooperates among themselves. The last group has 761 LP researchers, who neither coauthor with others within their own group nor collaborate strongly with researchers in other groups. They often write single author papers.

Table 15.1 Summaries with multiple resolutions for the DBLP DB coauthorship graph

Attribute Only	$k=4$	$k=5$	$k=6$	$k=7$
DB					

Now, in the *k-SNAP* summary for $k = 7$, we are curious if the average number of publications for each subgroup of the LP researchers is affected by the coauthorships with other groups. The above question can be easily answered by applying the *avg* operation on the *PubNum* attribute for each group in the result of the *k-SNAP* operation.

With this analysis, we find out that the group of LP researchers who collaborate with both P and HP researchers has a high average number of publications: 2.24. The group only collaborating with HP researchers has 1.66 publications on average. The group collaborating with only the P researchers has on average 1.55 publications. The group that tends to only cooperate among themselves has a low average number of publications: 1.26. Finally, the group of mostly single authors has on average only 1.23 publications. Not surprisingly, these results suggest that collaborating with HP and P researchers is potentially helpful for the low prolific (often beginning) researchers.

15.5 Scalability of the Graph Summarization Method

In this section, we take the top-down *k-SNAP* approach as an example to demonstrate the scalability of the graph summarization method described in this chapter, as it is more effective and efficient for most practical uses (see Section 15.3.3). More comprehensive experimental results can be found in [28].

Most real-world graphs show power-law degree distributions and small-world effect [20]. Therefore, the R-MAT model [9] in the GTgraph suites [2] is used to generate synthetic graphs with power-law degree distributions and small-world characteristics. The generator uses the default parameters values, and the average node degree in each synthetic graph is set to 5. An attribute is also assigned to each node in a generated graph. The domain of this attribute has five values. Each node is assigned randomly one of the five values.

The top-down *k-SNAP* approach was implemented in C++ on top of PostgreSQL(http://www.postgresql.org) and is applied to different sized synthetic graphs with three resolutions (k values): 10, 100, and 1000. This experiment was run on a 2.8 GHz Pentium 4 machine running Fedora 2, and equipped with a 250 GB SATA disk. 512 MB of memory is allocated to the PostgreSQL database buffer pool, and another 256 MB of additional memory is assigned as the working space outside the database system.

The execution times with increasing graph sizes are shown in Fig. 15.7. When $k = 10$, even on the largest graph with 1 million nodes and 2.5 million edges, the top-down *k-SNAP* approach finishes in about 5 min. For a given k value, the algorithm scales nicely with increasing graph sizes.

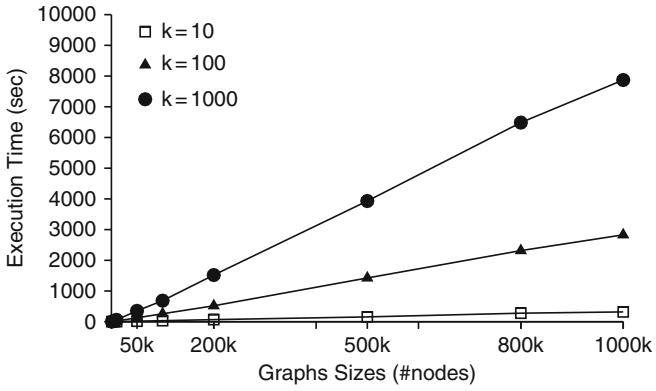


Fig. 15.7 Scalability experiment for synthetic data sets

15.6 Discussion

The example application and the experiments discussed above demonstrate the effectiveness and efficiency of the SNAP/k-SNAP summarization technique. However, the SNAP/k-SNAP approach described above has limitations, which we discuss below.

Attributes with Large Value Ranges: As can be seen from the algorithms in Section 15.3, the minimum summary size is bounded by the cardinality of the domain of the user-selected attributes. More precisely, the summary size is bounded by the actual number of distinct values in the graph data for the user-selected attributes. If one of the user-selected attributes has a large number of distinct values in the graph, then even the coarsest summary produced will be overwhelmingly large. One approach to solving this problem is to bucketize the attribute values into a small number of categories. One such approach is proposed in [30] to automatically categorize attributes with large distinct values by exploiting the domain knowledge hidden inside the node attributes values and graph link structures.

Large Number of Attributes: A related problem to the one discussed above is when a user selects a large number of node attributes for summarization. Now the attribute grouping in SNAP/K-SNAP has to be done over the cross-product of the distinct values used in each attribute domain. This cross-product space can be large. Bucketization methods can again be used in this case, though the problem is harder than the single attribute case. This is an interesting topic for future work.

15.7 Related Topics

15.7.1 Graph Compression

The graph summarization method introduced in this chapter produces small and informative summary graphs of the original graph. In some sense, these compact

summary graphs can be viewed as (lossy) compressed representations of the original graph, although the main goal of the summarization method we described is not to reduce the number of bits needed to encode the original graph, but enabling better understanding of the graph.

The related problem of graph compression has been extensively studied. Various compression techniques for unlabeled planar graphs have been proposed [10, 12, 15] and are generalized to graphs with constant genus [18]. In [5], a technique is proposed to compress unlabeled graphs based on the observation that most graphs in practice have small separators (subgraphs can be partitioned into two approximately equally sized parts by removing a relatively small number of vertices).

Due to the large scale of the web graph, a lot of attention has drawn to compress the web graph. Most of the studies have focused on lossless compression of the web graph so that the compact representation can be used to calculate measures such as PageRank [8]. Bharat et al. [4] proposed a compression technique making use of gaps between the nodes in the adjacency list. A reference encoding technique is introduced in [22], based on the observation that often a new web page adds links by copying links from an existing page. In this compression scheme, the adjacency list of one node is represented by referencing the adjacency list of another node. Alder and Mitzenmacher [1] proposed a minimum spanning tree-based algorithm to find the best reference list for the reference encoding scheme. The compression technique proposed in [27] takes advantage of the link structure of the web and achieves significant compression by distinguishing links based on whether they are inside or cross hosts, and by whether they are connecting popular pages or not. Boldi and Vigna [6, 7] developed a family of simple flat codes, called ζ codes, which are well suited for compressing power-law distributed data with small exponents. They achieve high edge compression and scale well to large graphs.

Two recently proposed graph compression techniques that share similarities with the graph summarization technique described in this chapter will be discussed in detail below.

15.7.1.1 S-Node Representation of the Web Graph

The compression technique proposed in [21] compresses the web graph into a *S-Node representation*. As exemplified in Fig. 15.8, the *S-Node representation* of a web graph contains the following components:

SUPERNODE GRAPH: The supernode graph is essentially a summary graph of the web graph, in which groups are called *supernodes* and group relationships are called *superedges*.

INTRANODE GRAPHS: Each intranode graph (abbreviated as IN in Fig. 15.8) characterizes the connections between the nodes inside a supernode.

POSITIVE SUPEREDGE GRAPHS: Each positive superedge graph (abbreviated as PSE in Fig. 15.8) is a directed bipartite graph that represents the links between two corresponding supernodes.

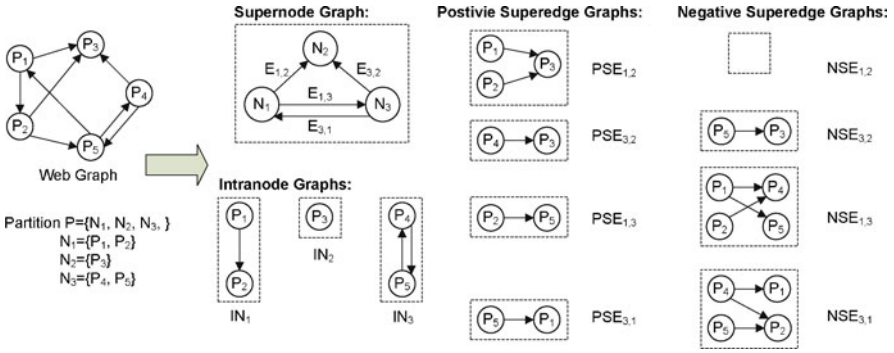


Fig. 15.8 An example of *S-Node representation* (from [21])

NEGATIVE SUPEREDGE GRAPHS: Each negative superedge graph (abbreviated as NSE in Fig. 15.8) captures, among all possible links between two supernodes, those that are absent from the actual web graph.

The compression technique in [21] employs a top-down approach to compute the S-Node representation. This algorithm starts from a set of supernodes that are generated based on the URL domain names, then iteratively splits an existing supernode by exploiting the URL patterns of the nodes inside this supernode and their links to other supernodes. However, different from the graph summarization method introduced in this chapter, this approach is specific to the web graph, thus are not directly applicable to other problem domains. Furthermore, since this approach aims at compressing the web graph, only one compressed S-Node representation is produced. Users have no control over the resolution of the summary graph.

15.7.1.2 MDL Representation of Graphs

Similar to the S-Node method described above, the technique proposed in [19] also compresses a graph into a *summary graph*. To reconstruct the original graph, a set of *edge corrections* are also produced. Figure 15.9 shows a sample graph G and its summary graph S with the set of edge corrections C.

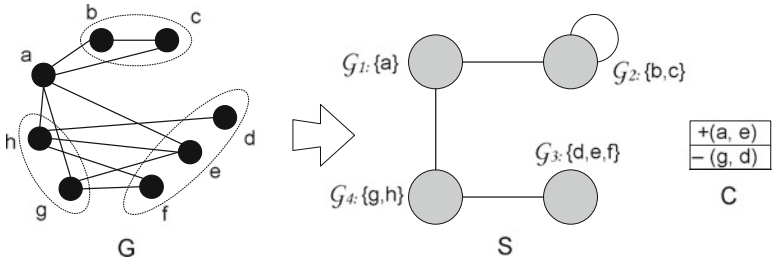


Fig. 15.9 An example of MDL-based summary: G is the original graph, S is the summary graph, and C is the set of edge corrections (from [25])

The original graph can be reconstructed from the summary graph by first adding an edge between each pair of nodes whose corresponding supernodes are connected by an superedge, then applying the edge corrections to remove non-existent edges from or add missing edges to the reconstructed graph. For example, to reconstruct the original graph in Fig. 15.9, the summary graph S is first expanded (now $V = \{a, b, c, d, e, f, g, h\}$ and $E = \{(a, b), (a, c), (a, h), (a, g), (b, c), (h, d), (h, e), (h, f), (g, d), (g, e), (g, f)\}$), then the set of corrections in C are applied: adding the edge (a, e) to E and removing the edge (g, d) from E.

Essentially, this proposed representation is equivalent to the S-Node representation described above. The intranode graphs, positive superedge graphs, and negative superedge graphs in the S-Node representation, collectively, can produce the edge corrections needed to reconstruct the original graph from the summary graph.

Based on Rissanen's minimum description length (MDL) principle [25], the authors in [19] formulated the graph compression problem into an optimization problem, which minimizes the sum of the size of the summary graph (the theory) and the size of the edge correction set (encoding of the original graph based on the theory). The representation with the minimum cost is called the *MDL representation*.

Two heuristic-based algorithms are proposed in [19] to compute the MDL representation of a graph. Both algorithms apply a bottom-up scheme: starting from the original graph and iteratively merging node pairs into supernodes until no further cost reduction can be achieved. The two algorithms differ in the policy of choosing which pair of nodes should merge in each iteration. The GREEDY algorithm always chooses the node pairs that give the maximum cost reduction, while the RANDOMIZED algorithm randomly picks a node and merges it with the best node in its vicinity.

The MDL representation can exactly reconstruct the original graph. However, for many applications, recreating the exact graph is not necessary. It is often adequate enough to construct a graph that is reasonably close to the original graph. As a result, the *ϵ-approximate MDL representation* is proposed to reconstruct the original graph within the user-specified bounded error ϵ ($0 \leq \epsilon \leq 1$).

To compute the ϵ-approximate MDL representation with the minimum cost, two heuristic-based algorithms are proposed in [19]. The first algorithm, called APXMDL, modifies the exact MDL representation by deleting corrections and summary edges while still satisfying the approximation constraint. The second algorithm, called APXGREEDY, incorporates the approximation constraint into the GREEDY algorithm, and constructs the ϵ-approximate representation directly from the original graph.

The key difference between this MDL-based approach and the *SNAP/k-SNAP* summarization approach is that the MDL method does not consider node attributes or multiple relationships in the summarization process and it does not allow users to control the resolutions of summaries.

15.7.2 Graph Visualization

Graph visualization methods are primarily designed to better layout a graph on a computer screen so that it is easier for users to understand the graph by visual inspection. Various graph drawing techniques are surveyed in [3]. However, as graphs become large, displaying an entire graph on the limited computer screen is challenging, both from the usability and the visual performance perspectives. To overcome the problems raised by the large graph sizes, navigation, interaction and, summarization techniques are often incorporated into graph visualization tools [13]. One common summarization technique used in graph visualization is structure-based clustering. Clustering provides abstraction of the original graph, and reduces the visual complexity. Graph visualization systems, such as [14, 24, 29], have applied clustering techniques to improve visualization clarity and at the same time increase performance of layout and rendering. The SuperGraph approach introduced in [26] employs a hierarchical graph partitioning technique to visualize large graphs in different resolution. In fact, the graph summarization technique introduced in this chapter can be coupled with visualization techniques to provide better understanding of large graphs.

15.8 Summary

This chapter studies an aggregation-based summarization method, which produces compact and informative graphs as the summaries of the original graphs. The summary graphs characterize the high-level structure embedded in the original graphs by aggregating nodes and edges from the original graph into node groups (supernodes) and group relationships (superedges), respectively. This summarization method utilizes the graph structure as well as user-specified node attributes and relationships to generate multi-resolution summaries. The users can interactively "drill-down" or "roll-up" to navigate through summaries with different resolution. Graph summarization is related to graph compression and can be coupled with graph visualization methods to enable better understanding of large graphs.

References

1. M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *Proceedings of the Data Compression Conference (DCC'01)*, page 203, IEEE Computer Society, Washington, DC, USA, 2001.
2. D. A. Bader and K. Madduri. GTgraph: A suite of synthetic graph generators. http://www.cc.gatech.edu/~kamesh/GTgraph.
3. G. Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Englewood Cliffs, NJ 1999.
4. K. Bharat, A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian. The connectivity server: Fast access to linkage information on the Web. *Computer Networks and ISDN Systems*, 30(1–7):469–477, 1998.

5. D. K. Blandford, G. E. Blelloch, and I. A. Kash. Compact representations of separable graphs. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 679–688, Baltimore, Maryland, USA, 2003.
6. P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of the International World Wide Web Conference (WWW'04)*, pages 595–602, New York, NY, USA, 2004.
7. P. Boldi and S. Vigna. The WebGraph framework II: Codes for the World-Wide Web. In *Proceedings of the Data Compression Conference (DCC'04)*, page 528, Snowbird, Utah, USA, 2004.
8. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the International World Wide Web Conference (WWW'98)*, pages 107–117, Amsterdam, The Netherlands, 1998.
9. D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *Proceedings of the SIAM International Conference on Data Mining (SDM'04)*, Lake Buena Vista, Florida, USA, 2004.
10. H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.
11. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'96)*, pages 152–159, New Orleans, Louisiana, USA, 1996.
12. X. He, M.-Y. Kao, and H.-I. Lu. A fast general methodology for information - theoretically optimal encodings of graphs. In *Proceedings of the European Symposium on Algorithms (ESA'99)*, pages 540–549, London, UK, 1999.
13. I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.
14. M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In *Proceedings of the International Symposium on Graph Drawing (GD'98)*, pages 374–383, London, UK, 1998.
15. K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3):239–252, 1995.
16. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. In *Proceedings of the International World Wide Web Conference (WWW'99)*, pages 1481–1493, Toronto, Canada, 1999.
17. M. Ley. DBLP Bibliography. http://www.informatik.uni-trier.de/~ley/db/.
18. H.-I. Lu. Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 223–224, San Francisco, California, USA, 2002.
19. S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, pages 419–432, Vancouver, Canada, 2008.
20. M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.
21. S. Raghavan and H. Garcia-Molina. Representing Web graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'03)*, pages 405–416, Bangalore, India, 2003.
22. K. H. Randall, R. Stata, R. G. Wickremesinghe, and J. L. Wiener. The link database: Fast access to graphs of the Web. In *Proceedings of the Data Compression Conference (DCC'02)*, pages 122–131, Washington, DC, USA, 2002.
23. D. G. Ravi, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'05)*, pages 721–732, Trondheim, Norway, 2005.

24. J. S. Risch, D. B. Rex, S. T. Dowson, T. B. Walters, R. A. May, and B. D. Moon. The STARLIGHT information visualization system. In *Proceedings of the IEEE Conference on Information Visualisation (IV'97)*, San Francisco, CA, USA, page 42, 1997.
25. J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.
26. J. F. Rodrigues, A. J. M. Traina, C. Faloutsos, and C. Traina. SuperGraph visualization. In *Proceedings of the IEEE International Symposium on Multimedia (ISM'06)*, Washington, DC, USA, 2006.
27. T. Suel and J. Yuan. Compressing the graph structure of the Web. In *Proceedings of the Data Compression Conference (DCC'01)*, pages 213–222, Washington, DC, USA, 2001.
28. Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, pages 567–580, Vancouver, Canada, 2008.
29. G. J. Wills. NicheWorks — interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics*, 8(2):190–212, 1999.
30. N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'10)*, Long Beach, California, USA, 2010.

Chapter 16
InfoNetOLAP: OLAP and Mining of Information Networks

Chen Chen, Feida Zhu, Xifeng Yan, Jiawei Han, Philip Yu, and Raghu Ramakrishnan

Abstract Databases and data warehouse systems have been evolving from handling normalized spreadsheets stored in relational databases to managing and analyzing diverse application-oriented data with complex interconnecting structures. Responding to this emerging trend, information networks have been growing rapidly and showing their critical importance in many applications, such as the analysis of XML, social networks, Web, biological data, multimedia data, and spatiotemporal data. Can we extend useful functions of databases and data warehouse systems to handle network structured data? In particular, OLAP (On-Line Analytical Processing) has been a popular tool for fast and user-friendly *multi-dimensional* analysis of data warehouses. *Can we OLAP information networks and perform mining tasks on top of that?* Unfortunately, to our best knowledge, there are no OLAP tools available that can interactively view and analyze network structured data from different perspectives and with multiple granularities. In this chapter, we argue that it is critically important to OLAP such information network data and propose a novel *InfoNetOLAP* framework. According to this framework, given an information network data set with its nodes and edges associated with respective attributes, a multi-dimensional model can be built to enable efficient on-line analytical processing so that *any portions* of the information networks can be *generalized/specialized* dynamically, offering multiple, versatile views of the data set. The contributions of this work are threefold. First, starting from basic definitions, i.e., what are *dimensions* and *measures* in the InfoNetOLAP scenario, we develop a conceptual framework for data cubes constructed on the information networks. We also look into different semantics of OLAP operations and classify the framework into two major subcases: *informational OLAP* and *topological OLAP*. Second, we show how an information network cube can be materialized by calculating a special kind of measure called *aggregated graph* and how to implement it efficiently. This includes both full materialization and partial materialization where constraints are enforced to obtain an *iceberg cube*. As we can see, due to the increased structural complexity

C. Chen (✉)
University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: cchen37@uiuc.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_16,

of data, aggregated graphs that depend on the underlying "graph" properties of the information networks are much harder to compute than their traditional OLAP counterparts. Third, to provide more flexible, interesting, and insightful OLAP of information networks, we further propose a *discovery-driven* multi-dimensional analysis model to ensure that OLAP is performed in an intelligent manner, guided by expert rules and knowledge discovery processes. We outline such a framework and discuss some challenging research issues for discovery-driven InfoNetOLAP.

16.1 Introduction

OLAP (On-Line Analytical Processing) [2, 7, 11, 12, 31] is an important notion in data analysis. Given the underlying data, a cube can be constructed to provide a *multi-dimensional* and *multi-level* view, which allows for effective analysis of the data from different perspectives and with multiple granularities. The key operations in an OLAP framework are slice/dice and roll-up/drill-down, with slice/dice focusing on a particular aspect of the data, roll-up performing generalization if users only want to see a concise overview, and drill-down performing specialization if more details are needed.

In a traditional data cube, a data record is associated with a set of dimensional values, whereas different records are viewed as *mutually independent*. Multiple records can be summarized by the definition of corresponding aggregate measures such as COUNT, SUM, and AVERAGE. Moreover, if a concept hierarchy is associated with each attribute, multi-level summaries can also be achieved. Users can navigate through different dimensions and multiple hierarchies via roll-up, drill-down, and slice/dice operations. However, in recent years, more and more data sources beyond conventional spreadsheets have come into being, such as chemical compounds or protein networks (chem/bio-informatics), 2D/3D objects (pattern recognition), circuits (computer-aided design), XML (data with loose schema), and Web (human/computer networks), where not only individual entities but also the *interacting relationships* among them are important and interesting. This demands a new generation of tools that can manage and analyze such data.

Given their great expressive power, information networks have been widely used for modeling a lot of data sets that contain structure information. With the tremendous amount of information network data accumulated in all above applications, the same need to deploy analysis from different perspectives and with multiple granularities exists. To this extent, our main task in this chapter is to develop an *InfoNetOLAP framework*, which presents a multi-dimensional and multi-level view over information networks.

In order to illustrate what we mean by "InfoNetOLAP" and how the OLAP glossary is interpreted with regard to this new scenario, let us start from a few examples.

Example 1 (Collaboration Patterns) There are a set of authors working in a given field: For any two persons, if they coauthor w papers in a conference, e.g.,

SIGMOD 2004, then a link is added between them, which has a collaboration frequency attribute that is weighted as w. For every conference in every year, we may have a coauthor network describing the collaboration patterns among researchers; each of them can be viewed as a snapshot of the overall coauthor network in a bigger context.

It is interesting to analyze the aforementioned network data set in an OLAP manner. First, one may want to check the collaboration patterns for a group of conferences, say, all DB conferences in 2004 (including SIGMOD, VLDB, ICDE, etc.) or all SIGMOD conferences since the year it was introduced. In the language of data cube, with a *venue* dimension and a *time* dimension, one may choose to obtain the *(db-conf, 2004)* cell and the *(sigmod, all-years)* cell, where the *venue* and *time* dimensions have been generalized to *db-conf* and *all-years*, respectively. Second, for the subset of snapshots within each cell, one can summarize them by computing a measure as we did in traditional OLAP. In the InfoNetOLAP context, this gives rise to an *aggregated graph*. For example, a summary network displaying total collaboration frequencies can be achieved by overlaying all snapshots together and summing up the respective edge weights, so that each link now indicates two persons' collaboration activities in the DB conferences of 2004 or during the whole history of SIGMOD. ■

The above example is simple because the measure is calculated by a simple sum over individual pieces of information. A more complex case is presented next.

Example 2 (Maximum Flow) Consider a set of cities connected by transportation networks. In general, there are many ways to go from one city A to another city B, e.g., by bus, by train, by air, by water, and each is operated by multiple companies. For example, we can assume that the capacity of company x's air service from A to B is c^x_{AB}, i.e., company x can transport at most c^x_{AB} units of cargo from A to B using the planes it owns. Finally, we get a snapshot of capacity network for every transportation means of every company.

Now, consider the transporting capability from a source city S to a destination city T; it is interesting to see how this value can be achieved by sending flows of cargo through different paths if (1) we only want to go by air or (2) we only want to choose services operated by company x. In the OLAP language, with a *transportation-type* dimension and a *company* dimension, the above situations correspond to the *(air, all-companies)* cell and the *(all-types, company x)* cell, while the *measure* computed for a cell c is a graph displaying how the maximum flow can be configured, which has considered all transportation means and operating companies associated with c. Unlike Example 1, computing the aggregated graph of maximum flow is now a much harder task; also, the semantics associated with un-aggregated network snapshots and aggregated graphs are different: The former shows capacities on its edges, whereas the latter shows transmitted flows, which by definition must be smaller. ■

Example 3 (Collaboration Patterns, Revisited) Usually, the whole coauthor network could be too big for the users to comprehend, and thus it is desirable to look at a

more compressed view. For example, one may like to see the collaboration activities organized by the authors' associated affiliations, which requires the network to be generalized one step up, i.e., merging all persons in the same institution as one node and constructing a new summary graph at the institution level. In this "generalized network", for example, an edge between Stanford and University of Wisconsin will aggregate all collaboration frequencies incurred between Stanford authors and Wisconsin authors. Similar to Examples 1 and 2, an aggregated graph (i.e., the generalized network defined above) is taken as the OLAP measure. However, the difference here is that a roll-up from the individual level to the institution level is achieved by consolidating multiple nodes into one, which shrinks the original network. Compared to this, the graph in Examples 1 and 2 is not collapsed because we are always examining the relationships among the same set of objects; it poses minimum changes with regard to network topology upon generalization. ■

The above examples demonstrate that OLAP provides a powerful primitive to examine information networks. In this chapter, we will give a systematic study on InfoNetOLAP, which is more general than the traditional OLAP: In addition to individual entities, the mutual interactions among them are also considered in the analytical process. Our major contributions are summarized below.

1. *On conceptual modeling*, an *InfoNetOLAP framework* is developed, which defines *dimensions* and *measures* in the information network context, as well as the concept of *multi-dimensional* and *multi-level* analysis over network structured data. We distinguish different semantics of OLAP operations and categorize them into two major subcases: *informational OLAP* (as shown in Examples 1 and 2) and *topological OLAP* (as shown in Example 3). It is necessary since these two kinds of OLAP demonstrate substantial differences with regard to the construction of data cubes over information networks.
2. *On efficient implementation*, the computation of *aggregated graphs* as InfoNetOLAP measures is examined. Due to the increased structural complexity of data, calculating certain measures that are closely tied with the "graph" properties of the information networks, e.g., maximum flow, centrality, poses greater challenges than their traditional OLAP counterparts, such as COUNT, SUM, and AVERAGE. We investigate this issue, categorize measures based on the difficulty to compute them in the OLAP context, and suggest a few measure properties that might help further optimize processings. Both full materialization and partial materialization (where constraints are enforced to obtain an *iceberg cube*) are discussed.
3. *On utilizing the framework for knowledge discovery*, we propose *discovery-driven InfoNetOLAP*, so that interesting patterns and knowledge can be effectively discovered. After presenting the general ideas, we outline a list of guiding principles and discuss some associated research problems. Being part of an ongoing project, it opens a promising path that could potentially lead to more flexible and insightful OLAP of information networks, which we shall explore further in future works.

The remainder of this chapter is organized as follows. In Section 16.2, we formally introduce the InfoNetOLAP framework. Section 16.3 discusses the general hardness to compute aggregated graphs as InfoNetOLAP measures, which categorizes them into three classes. Section 16.4 looks into some properties of the measures and proposes a few computational optimizations. Constraints and partial materialization are studied in Section 16.5. Discovery-driven InfoNetOLAP is covered in Section 16.6. We report experiment results and related work in Sections 18.7 and 16.8, respectively. Section 16.9 concludes this study.

16.2 An InfoNetOLAP Framework

In this section, we present the general framework of InfoNetOLAP.

Definition 1 (Information Network Model) We model the data examined by InfoNetOLAP as *a collection of network snapshots* $\mathcal{G} = \{\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_N\}$, where each snapshot $\mathbb{G}_i = (I_{1,i}, I_{2,i}, \ldots, I_{k,i}; G_i)$ in which $I_{1,i}, I_{2,i}, \ldots, I_{k,i}$ are k *informational attributes* describing the snapshot as a whole and $G_i = (V_i, E_i)$ is a *graph*. There are also *node attributes* attached with any $v \in V_i$ and *edge attributes* attached with any $e \in E_i$. Note that since $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_N$ only represent different observations, $V_1, V_2, \ldots, V_N$ actually correspond to the same set of objects in real applications.

For instance, with regard to the coauthor network described in the introduction, *venue* and *time* are two informational attributes that mark the status of individual snapshots, e.g., SIGMOD 2004 and ICDE 2005, *authorID* is a node attribute indicating the identification of each node, and *collaboration frequency* is an edge attribute reflecting the connection strength of each edge.

Dimension and *measure* are two concepts that lay the foundation of OLAP and cubes. As their names imply, first, dimensions are used to construct a cuboid lattice and partition the data into different cells, which act as the basis for multi-dimensional and multi-level analysis; second, measures are calculated to aggregate the data covered, which deliver a summarized view of it. Below, we are going to formally re-define these two concepts concerning the InfoNetOLAP scenario.

Let us examine dimensions at first. Actually, there are two types of dimensions in InfoNetOLAP. The first one, as exemplified by Example 1, utilizes informational attributes attached at the whole snapshot level. Suppose the following concept hierarchies are associated with *venue* and *time*:

- *venue*: *conference* → *area* → *all*,
- *time*: *year* → *decade* → *all*;

the role of these two dimensions is to organize snapshots into groups based on different perspectives, e.g., *(db-conf, 2004)* and *(sigmod, all-years)*, where each of these groups corresponds to a "cell" in the OLAP terminology. They control what snapshots are to be looked at, without touching the inside of any single snapshot.

Definition 2 (Informational Dimensions) With regard to the information network model presented in Definition 1, the set of informational attributes $\{I_1, I_2, \ldots, I_k\}$ are called the *informational dimensions* of InfoNetOLAP, or *Info-Dims* in short.

The second type of dimensions is provided to operate on nodes and edges within individual networks. Take Example 3 for instance; suppose the following concept hierarchy

- *authorID*: *individual* → *department* → *institution* → *all*

is associated with the node attribute *authorID*, then it can be used to group authors from the same institution into a "generalized" node, and a new network thus formed will depict interactions among these groups as a whole, which summarizes the original network and hides specific details.

Definition 3 (Topological Dimensions) The set of dimensions coming from the attributes of topological elements (i.e., nodes and edges of G_i), $\{T_1, T_2, \ldots, T_l\}$, are called the *topological dimensions* of InfoNetOLAP, or *Topo-Dims* in short.

The OLAP semantics accomplished through Info-Dims and Topo-Dims are rather different, and in the following we shall refer to them as *informational OLAP* (abbr. *I-OLAP*) and *topological OLAP* (abbr. *T-OLAP*), respectively.

For roll-up in *I-OLAP*, the characterizing feature is that snapshots are just different observations of the same underlying network, and thus when they are all grouped into one cell in the cube, it is like *overlaying* multiple pieces of information, *without changing* the objects whose interactions are being looked at.

For roll-up in *T-OLAP*, we are no longer grouping snapshots, and the reorganization switches to happen inside individual networks. Here, *merging* is performed internally which "zooms out" the user's focus to a "generalized" set of objects, and a new information network formed by such *shrinking* might greatly alter the original network's topological structure.

Now we move on to measures. Remember that, in traditional OLAP, a measure is calculated by aggregating all the data tuples whose dimensions are of the same values (based on concept hierarchies, such values could range from the finest ungeneralized ones to "all/*", which form a multi-level cuboid lattice); casting this to our scenario here:

First, in InfoNetOLAP, the aggregation of graphs should also take the form of a graph, i.e., an *aggregated graph*. In this sense, graph can be viewed as a special existence, which plays a dual role: as a data source and as an aggregated measure. Of course, other measures that are not graphs, such as node count, average degree, diameter, can also be calculated; however, we do not explicitly include such non-graph measures in our model, but instead treat them as derived from corresponding graph measures.

Second, due to the different semantics of I-OLAP and T-OLAP, aggregating data with identical Info-Dim values groups information among the snapshots, whereas aggregating data with identical Topo-Dim values groups topological elements inside individual networks. As a result, we will give a separate measure definition for each case below.

Definition 4 (I-Aggregated Graph) With regard to Info-Dims $\{I_1, I_2, \ldots, I_k\}$, the *I-aggregated graph* M^I is an *attributed graph* that can be computed based on a set of network snapshots $\mathcal{G}' = \{\mathbb{G}_{i_1}, \mathbb{G}_{i_2}, \ldots, \mathbb{G}_{i_{N'}}\}$ whose Info-Dims are of identical values; it satisfies the following: (1) the nodes of M^I are as same as any snapshot in $\mathcal{G}'$, and (2) the node/edge attributes attached to M^I are calculated as *aggregate functions* of the node/edge attributes attached to $G_{i_1}, G_{i_2}, \ldots, G_{i_{N'}}$.

The graph in Fig. 16.1 that describes collaboration frequencies among individual authors for a particular group of conferences during a particular period of time is an instance of I-aggregated graph, and the interpretation of classic OLAP operations with regard to InfoNet I-OLAP is summarized as follows:

- Roll-up: Overlay multiple snapshots to form a higher-level summary via I-aggregated graph.
- Drill-down: Return to the set of lower-level snapshots from the higher-level overlaid (aggregated) graph.
- Slice/dice: Select a subset of qualifying snapshots based on Info-Dims.

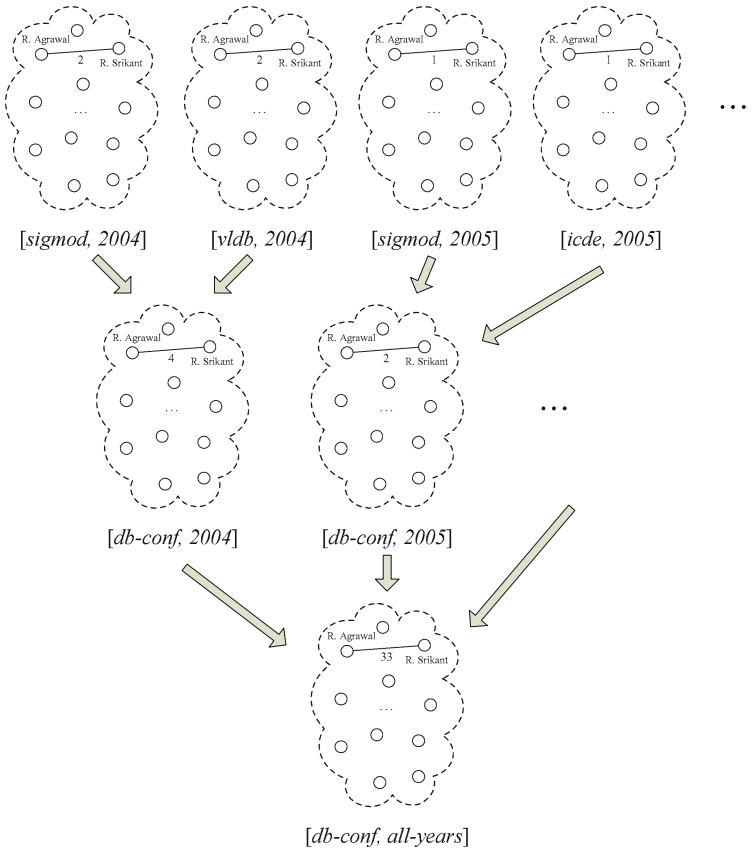


Fig. 16.1 The OLAP scenario for Example 1

Definition 5 (T-Aggregated Graph) With regard to Topo-Dims $\{T_1, T_2 \ldots, T_l\}$, the *T-aggregated graph* M^T is an *attributed graph* that can be computed based on an individual network G_i; it satisfies the following: (1) the nodes in G_i that have identical values on their Topo-Dims are grouped, whereas each group corresponds to a node of M^T, and (2) the attributes attached to M^T are calculated as *aggregate functions* of the attributes attached to G_i.

The graph in Fig. 16.2 that describes collaboration frequencies among institutions is an instance of T-aggregated graph, and the interpretation of classic OLAP operations with regard to InfoNet T-OLAP is summarized as follows:

- Roll-up: Shrink the network topology and obtain a T-aggregated graph that displays compressed views. Topological elements (i.e., nodes and/or edges) are merged and replaced by corresponding higher-level ones during the process.
- Drill-down: A reverse operation of roll-up.
- Slice/dice: Select a subgraph of the network based on Topo-Dims.

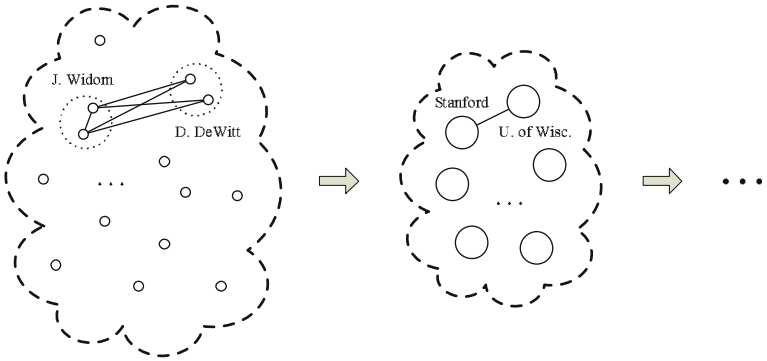


Fig. 16.2 The OLAP scenario for Example 3

16.3 Measure Classification

Now, with a clear concept of dimension, measure, and possible OLAP operations, we are ready to discuss implementation issues, i.e., how to compute the aggregated graph in a multi-dimensional and multi-level way.

Recall that in traditional OLAP, measures can be classified into *distributive*, *algebraic*, and *holistic*, depending on whether the measures of high-level cells can be easily computed from their low-level counterparts, without accessing base tuples residing at the finest level. For instance, in the classic *sale*(*time*, *location*) example, the total sale of [*2008*, *California*] can be calculated by adding up the total sales of [*January 2008*, *California*], [*February 2008*, *California*], …, [*December 2008*, *California*], without looking at base data points such as [*04/12/2008*, *Los Angeles*], which means that SUM is a distributive measure. Compared to this, AVG has been

used to illustrate algebraic measures, which is actually a *semi-distributive* category in that AVG can be derived from two distributive measures: SUM and COUNT, i.e., algebraic measures are functions of distributive measures.

(Semi-)distributiveness is a nice property for top-down cube computation, where the cuboid lattice can be gradually filled up by making level-by-level aggregations. Measures without this property is put into the holistic category, which is intuitively much harder to calculate. Now, concerning InfoNetOLAP, based on similar criteria with regard to the aggregated graphs, we can also classify them into three categories.

Definition 6 (Distributive, Algebraic, and Holistic) Consider a high-level cell c_h and the corresponding low-level cells it covers: $c_l^1, c_l^2, \ldots$. An aggregated graph M_d is *distributive* if $M_d(c_h)$ can be directly computed as a function of $M_d(c_l^1), M_d(c_l^2), \ldots$, i.e.,

$$M_d(c_h) = F_d\big[M_d(c_l^1), M_d(c_l^2), \ldots\big].$$

For a non-distributive aggregated graph M_a, if it can be derived from some other distributive aggregated graphs $M_d^1, M_d^2, \ldots$, i.e., for $\forall c_h$,

$$M_a(c_h) = F_a\big[M_d^1(c_h), M_d^2(c_h), \ldots\big],$$

then we say that it is *algebraic*. Aggregated graphs that are neither distributive nor algebraic belong to the *holistic* category.

If we further distinguish between I-OLAP and T-OLAP, there are actually six classes: I-distributive, I-algebraic, I-holistic and T-distributive, T-algebraic, T-holistic. Because of their wide differences with regard to semantics as well as implementation, let us first focus on InfoNet I-OLAP.

I-Distributive. The I-aggregated graph describing collaboration frequency in Example 1 is an I-distributive measure, because the frequency value in a high-level I-aggregated graph can be calculated by simply adding those in corresponding low-level I-aggregated graphs.

I-Algebraic. The graph displaying maximum flow configuration in Example 2 is an I-algebraic measure based on the following reasoning. Suppose *transportation type* and *company* comprise the two dimensions of a cube, and we generalize from low-level cells *(all-types, company 1)*, *(all-types, company 2)*, ... to *(all-types, all-companies)*, i.e., compute the maximum flow based on all types of transportation operated by all companies. Intuitively, this overall maximum flow would not be a simple sum (or other indirect manipulations) of each company's individual maximum flows. For instance, company 1 may have excessive transporting capability between two cities, whereas the same link happens to be a bottleneck for company 2: Considering both companies together for determination of the maximum flow can enable capacity sharing and thus create a double-win situation. In this sense, maximum flow is not distributive by definition. However, as an obvious fact, maximum flow f is decided by the network c that shows link capacities on its edges, and this capacity graph is distributive because it can be directly added upon generalization:

When link sharing is enabled, two separated links from A to B, which are operated by different companies and have capacities c^1_{AB} and c^2_{AB}, respectively, are no different from a single link with capacity $c^1_{AB} + c^2_{AB}$, considering their contributions to the flow value. Finally, being a function of distributive aggregated graphs, maximum flow is algebraic.

I-holistic. The I-holistic case involves a more complex aggregated graph, where base-level details are required to compute it. In the coauthor network of Example 1, the median of researchers' collaboration frequency for all DB conferences from 1990 to 1999 is holistic, similar to what we have seen in a traditional data cube.

Based on the same classification criteria, considering InfoNet T-OLAP, *T-distributive*, *T-algebraic*, and *T-holistic* aggregated graphs also exist. As an example, we will prove below that the measure graph showing degree centralities of vertices in a social network G is T-algebraic, where the *degree centrality* $C_D(v)$ of a vertex v is equivalent to the number of edges that connect v to other vertices in G.

Suppose we are performing topological roll-up operations on the coauthor network as we did in Fig. 16.2, and we are going from the individual level to the department level and then to the institution level. Here, we count each coauthored publication as an edge; so, if the collaboration frequency of two persons is 4, it means that there are four edges. Now, consider a set of vertices $V = \{v_1, v_2, \ldots, v_s\}$ that belong to the same node group and thus will be merged together into a single vertex v' by T-OLAP operations; it is easy to verify that

$$C_D(v') = \sum_{1 \le i \le s} C_D(v_i) - 2|E_V|,$$

where E_V is the number of edges that have both their ends in V. Figure 16.3 is an illustration of the computation process. In G'', it can be seen that $C_D(v')$ is a total of 4, 2, 5, and 3, which is 14. We can get this number directly from the original network G by the given formula $\sum_{1 \le i \le s} C_D(v_i) - 2|E_V| = (3+8+3+7+10+11+7+5+6) - 2(2+3+3+1+2+4+1+2+4+1) = 14$, while we can also leverage results obtained from the intermediate network G': $(8+12+14) - 2(3+1+6) = 14$. Since the computational cost is $O(s + |E_V|)$, where $s = |V|$ is the number of vertices in V, this example demonstrates that the computation cost can be greatly reduced by following the top-down materialization order and taking advantage of results that are already computed along the way, as G' has much less vertices than G.

16.4 Optimizations

Being (semi-)distributive or holistic tells us whether the aggregated graph computation needs to start from completely un-aggregated data or some intermediate results can be leveraged. However, even if the aggregated graph is distributive or algebraic, and thus we can calculate high-level measures based on some intermediate-level

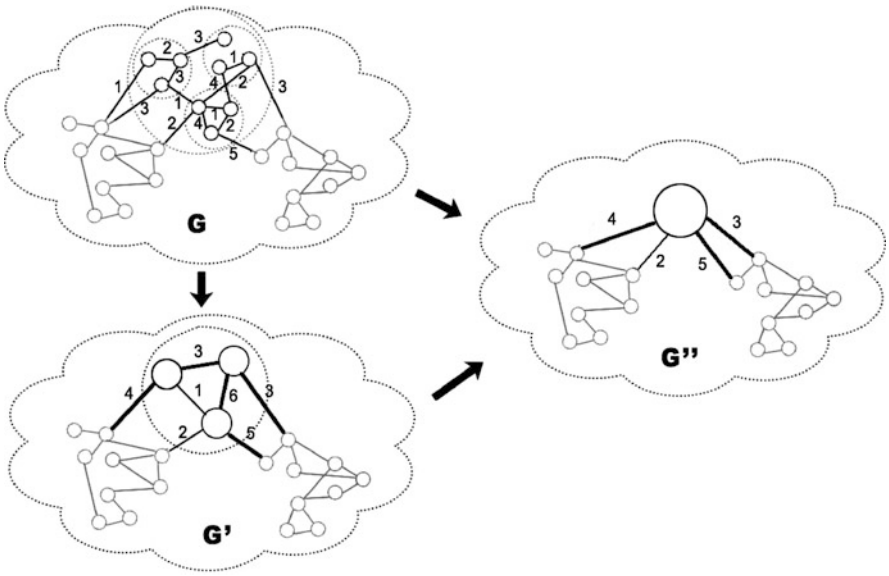


Fig. 16.3 Computing degree centrality

ones, it is far from enough, because the complexity to compute the two functions F_d and F_a in Definition 6 is another question. Think about the maximum flow example we just mentioned; F_a takes the distributive capacity graph as input to compute the flow configuration, which is by no means an easy transformation.

Based on our analysis, there are mainly two reasons for such potential difficulties. First, due to the interconnecting nature of graphs, the computation of many information network properties is "global" as it requires us to take the whole network into consideration. In order to make this concept of globalness clear, let us first look at a "local" situation: For I-OLAP, aggregated graphs are built for the same set of objects as the underlying network snapshots; now, in the aggregated graph of Example 1, "R. Agrawal" and "R. Srikant"'s collaboration frequency for cell *(db-conf, 2004)* is locally determined by "R. Agrawal" and "R. Srikant"'s collaboration frequency for each of the DB conferences held in 2004; it does not need any information from the other authors to fulfill the computation. This is an ideal scenario, because the calculations can be localized and thus greatly simplified. Unfortunately, not all measures provide such local properties. For instance, in order to calculate a maximum flow from S to T for the cell *(air, all-companies)* in Example 2, only knowing the transporting capability of each company's direct flights between S and T is not enough, because we can always take an indirect route via some other cities to reach the destination.

Second, the purpose for us to compute high-level aggregated graphs based on low-level ones is to reuse the intermediate calculations that are already performed. Taking I-OLAP for example, when computing the aggregated graph of a low-level cell c_l^i $(i = 1, 2, \ldots)$, we only had a partial view about the c_l^i-portion of network

snapshots; now, as multiple pieces of information are overlaid into the high-level cell c_h, some full-scale consolidation needs to be performed, which is very much like the merge sort procedure, where partial ranked lists are reused but somehow adjusted to form a full ranked list. Still, because of the structural complexity, it is not an easy task to develop such reuse schemes for information networks, and even if it is possible, reasonably complicated operations might be involved. Similar problems exist for InfoNet T-OLAP as well.

Admitting the difficulties above, let us now investigate the possibility to alleviate them. As we have seen, for some aggregated graphs, the first aspect can be helped by their *localization* properties with regard to network topology. Concerning the second aspect, the key is how to effectively reuse partial results computed for intermediate cells so that the workload to obtain a full-scale measure is *attenuated* as much as possible. In the following, we are going to examine these two directions in sequel.

16.4.1 Localization

Definition 7 (Localization in InfoNet I-OLAP) For an I-aggregated graph M_l^I that summarizes a group of network snapshots $\mathcal{G}' = \{\mathbb{G}_{i_1}, \mathbb{G}_{i_2}, \ldots, \mathbb{G}_{i_{N'}}\}$, if (1) we only need to check a *neighborhood* of v in $G_{i_1}, G_{i_2}, \ldots, G_{i_{N'}}$ to calculate v's node attributes in M_l^I, and (2) we only need to check a *neighborhood* of u, v in $G_{i_1}, G_{i_2}, \ldots, G_{i_{N'}}$ to calculate (u, v)'s edge attributes in M_l^I, then the computation of M_l^I is said to be *I-OLAP localizable*.

Example 4 (Common Friends) With regard to the coauthor network depicted in Example 1, we can also compute the following aggregated graph: Given two authors a_1 and a_2, the edge between them records the number of their common "friends", whereas in order to build such "friendship", the total collaboration frequency between two researchers must surpass a δ_c threshold for the specified conferences and time. ■

The above example provides another instance that leverages localization to promote efficient processing. Consider a cell, e.g., *(db-conf, 2004)*, the determination of the aggregated graph's a_1-a_2 edge can be restricted to a 1-neighborhood of these two authors in the un-aggregated snapshots of 2004's DB conferences, i.e., we only need to check edges that are directly adjacent to either a_1 or a_2, and in this way a third person a_3 can be found, if he/she publishes with both a_1 and a_2, while the total collaboration frequency summed from the weights of these adjacent edges is at least δ_c. Also, note that the above aggregated graph definition is based on the number of length-2 paths like a_1-a_3-a_2 where each edge of it represents a "friendship" relation; now, if we further allow the path length to be at most k, computations can still be localized in a $\lfloor \frac{k}{2} \rfloor$-neighborhood of both authors, i.e., any relevant author on such paths of "friendship" must be reachable from either a_1 or a_2 within $\lfloor \frac{k}{2} \rfloor$ steps. This can be seen as a situation that sits in the middle of Example 1's "absolute locality" (0-neighborhood) and maximum flow's "absolute globality" (∞-neighborhood).

There is an interesting note we want to put for the *absolutely local distributive* aggregated graph of Example 1. Actually, such a 0-neighborhood localization property degenerates the scenario to a very special case, where it is no longer necessary to assume the underlying data as an information network: For each pair of coauthors, we can construct a traditional cube showing their collaboration frequency "OLAPed" with regard to *venue* and *time*, whose computation does not depend on anything else in the coauthor network. In this sense, we can treat the coauthor network as a union of pairwise collaboration activities, whereas Example 1 can indeed be thought as a traditional OLAP scenario disguised under its information network appearances, because the cube we constructed here is nothing different from a collection of pairwise traditional cubes. As a desirable side effect, this enables us to leverage specialized technologies that are developed for traditional OLAP, which in general could be more efficient. Nevertheless, the case is special, anyway: Absolute localization would not hold for most information network measures, which is also the reason why traditional OLAP proves to be extremely restricted when handling network data.

The corresponding definition of localization in InfoNet T-OLAP is given as follows, and it can be easily proved that the degree centrality we have discussed above is 1-neighborhood T-OLAP localizable.

Definition 8 (Localization in InfoNet T-OLAP) For a T-aggregated graph M_l^T where a vertex v' of it represents a group of nodes $V = \{v_1, v_2, \ldots, v_s\}$ of the original network G_i, if (1) we only need to check a *neighborhood* of $v_1, v_2, \ldots$ in G_i to calculate v''s node attributes in M_l^T, and (2) we only need to check a *neighborhood* of $u_1, u_2, \ldots$ and $v_1, v_2, \ldots$ in G_i to calculate (u', v')'s edge attributes in M_l^T, then the computation of M_l^T is said to be *T-OLAP localizable*.

16.4.2 Attenuation

Below, we are going to explain the idea of attenuation through examples, and the case we pick is maximum flow. In a word, *the more partial results from intermediate calculations are utilized, the more we can decrease the cost of obtaining a full-scale aggregated graph*.

To begin with, let us first review some basic concepts, cf. [8]. Given a directed graph $G = (V, E)$, $c : \binom{V}{2} \to R^{\geq 0}$ indicates a *capacity* for all pairs of vertices and E is precisely the set of vertex pairs for which $c > 0$. For a source node s and a destination node t, a *flow* in G is a function $f : \binom{V}{2} \to R$ assigning values to graph edges such that (i) $f(u, v) = -f(v, u)$: skew symmetry, (ii) $f(u, v) \leq c(u, v)$: capacity constraint, and (iii) for each $v \neq s/t$, $\sum_{u \in V} f(u, v) = 0$: flow conservation. Since most maximum flow algorithms work incrementally, there is an important lemma as follows.

Lemma 1 *Let f be a flow in G and let G_f be its residual graph, where* residual *means that the capacity function of G_f is $c_f = c - f$; now, f' is a maximum flow in G_f if and only if $f + f'$ is a maximum flow in G.*

Note that the $+/-$ notation here means edge-by-edge addition/subtraction; and in summary, this lemma's core idea is to look for a flow f' in G_f and use f' to *augment* the current flow f in G.

For the InfoNetOLAP context we consider, in order to compute the algebraic aggregated graph displaying maximum flow, the function F_a takes a distributive capacity graph c as its input; now, since capacity can be written as the sum of a flow and a residual graph: $c = f + c_f$, does this decomposition provide us some hints to pull out the useful part f, instead of blindly taking c and starting from scratch?

Suppose that the capacity graph of cell *(all-types, company 1)* is c^1, where f_1 is the maximum flow and $c^1_{f_1} = c^1 - f_1$ denotes the corresponding residual graph. Likewise, we have c^2, f_2, and $c^2_{f_2}$ for cell *(all-types, company 2)*. Without loss of generality, assume there are only these two companies whose transportation networks are overlaid into *(all-types, all-companies)*, which has a capacity of $c = c^1 + c^2$.

Claim $f_1 + f_2 + f'$ is a maximum flow for c if and only if f' is a maximum flow for $c^1_{f_1} + c^2_{f_2}$.

Proof Since f_1 and f_2 are restricted to the transportation networks of company 1 and company 2, respectively, the overall capacity $c = c^1 + c^2$ must accommodate $f_1 + f_2$, even if link sharing is not enabled. As a result of subtracting $f_1 + f_2$, the residual graph becomes

$$\begin{aligned} c_{f_1+f_2} &= (c^1 + c^2) - (f^1 + f^2) \\ &= (c^1 - f^1) + (c^2 - f^2) = c^1_{f_1} + c^2_{f_2}. \end{aligned}$$

A direct application of Lemma 1 finishes our proof. ■

As it is generally hard to localize maximum flow computations with regard to network topology, the above property is important because it takes another route, which reuses partial results f_1, f_2 and attenuates the overall workload from c^1+c^2 to $c^1_{f_1}+c^2_{f_2}$. By doing this, we are much closer to the overall maximum flow f_1+f_2+f' because a big portion of it, $f_1 + f_2$, has already been decided even before we start an augmenting algorithm.

However, we should admit that attenuation schemes usually take widely different forms, which might need to be developed with regard to specific aggregate measure graphs; furthermore, as we shall see next, there do exist cases where such properties are hard, if not impossible, to think of.

Example 5 (Centrality) *Centrality* is an important concept in social network analysis, which reflects how "central" a particular node's position is in a given network. One definition called *betweenness centrality* C_B uses shortest path to model this: Let n_{jk} denote the number of shortest paths (as there could be equally short ones) between two nodes j and k; for any node i, $\frac{n_{jk}(i)}{n_{jk}}$ is the fraction of shortest paths between j, k that go through i, with $C_B(i)$ summing it up over all possible pairs: $C_B(i) = \sum_{j,k \neq i} \frac{n_{jk}(i)}{n_{jk}}$. Intuitively, for a "star"-shaped network, all shortest

paths must pass the network center, which makes C_B achieve its maximum value $(|V|-1)(|V|-2)/2$.

Only considering shortest paths is inevitably restrictive in many situations; and thus, *information centrality* C_I goes one step further by taking all paths into account. It models any path from j to k as a signal transmission, which has a channel noise proportional to its path length. For more details, we refer the readers to [25], which has derived the following formula based on information theoretic analysis: Let A be a matrix, whose a_{ij} entry designates the interaction strength between node i and node j; define $B = D - A + J$, where D is a diagonal matrix with $D_{ii} = \sum_{j=1}^{|V|} a_{ij}$ and J is a matrix having all unit elements; now, perform an inverse operation to get the *centrality matrix* $C = B^{-1}$; write its diagonal sum as $T = \sum_{j=1}^{|V|} c_{jj}$ and its row sum as $R_i = \sum_{j=1}^{|V|} c_{ij}$; the information centrality of node i is then equivalent to

$$C_I(i) = \frac{1}{c_{ii} + (T - 2R_i)/|V|}.$$

Now, with regard to the coauthor network described in Example 1, if we define the interaction strength between two authors as their total collaboration frequency for a set of network snapshots, then an aggregated graph M_{cen} can be defined, whose node i is associated with a node attribute $C_I(i)$ equivalent to its information centrality. ■

Claim The computation of M_{cen} is hard to be attenuated in a level-by-level aggregation scheme.

Proof As we can see, the core component of information centrality computation is a matrix inverse. Now, given two portions of network snapshots that are overlaid, the overall centrality matrix is

$$\big[(D_1 + D_2) - (A_1 + A_2) + J\big]^{-1} = (B_1 + B_2 - J)^{-1}.$$

From calculations performed on lower levels, we know the centrality matrices $C_1 = B_1^{-1}$ and $C_2 = B_2^{-1}$; however, it seems that they do not help much to decrease the computation cost of inverting $B_1 + B_2 - J$. ■

When things like this happen, an alternative is to abandon the exactness requirement and use intermediate results that are readily available to bound the answer within some range instead; as we shall elaborate in the following section, this will become very useful if the cube construction is subject to a set of constraints.

16.5 Constraints and Partial Materialization

Above, we have focused on the computation of a full cube, i.e., each cell in each cuboid is calculated and stored. In many cases, this is too costly in terms of both

space and time, which might even be unnecessary if the users are not interested in obtaining all the information. Usually, users may stick with an *interestingness* function I, indicating that only those cells above a particular threshold δ make sense to them. Considering this, all cells c with $I(c) \geq \delta$ comprise an *iceberg* cube, which represents a *partial materialization* of the cube's interesting part. Taking Example 2 for instance, it is possible that people may only care about those subnetworks that can transmit at least $\delta_{|f|}$ units of cargo, while the other cells are discarded from consideration, due to their limited usefulness for the overall transportation business.

Optimizations exist as to how such an iceberg cube can be calculated, i.e., how to efficiently process constraints like $I(c) \geq \delta$ during materialization, without generating a full cube at first. Below, we will first classify different constraints into categories (Section 16.5.1) and then combine with some examples to see how each category should be dealt with (Section 16.5.2).

16.5.1 Constraint Classification

Two most important categories of constraints are *anti-monotone* and *monotone*. They relate cells on different levels of the cube together and are defined as follows.

Definition 9 A constraint C is *anti-monotone*, if for any high-level cell c_h and a low-level cell c_l covered by c_h, the following must hold: c_h violates $C \Rightarrow c_l$ violates C.

Definition 10 A constraint C is *monotone*, if for any high-level cell c_h and a low-level cell c_l covered by c_h, the following must hold: c_h satisfies $C \Rightarrow c_l$ satisfies C.

Note that, in Definition 9, "c_h violates $C \Rightarrow c_l$ violates C" is equal to "c_l satisfies $C \Rightarrow c_h$ satisfies C" and, in Definition 10, "c_h satisfies $C \Rightarrow c_l$ satisfies C" is equal to "c_l violates $C \Rightarrow c_h$ violates C"; so, depending on whether c_h or c_l is computed (and thus verified against C) first, there are different ways to make use of these constraints, which we will demonstrate below.

16.5.2 Constraint Pushing

The anti-monotone and monotone constraints can be "pushed" deep into the computation process using the Apriori principle [30]. In general, there are two approaches to compute a data cube over information networks, *bottom-up* and *top-down*. In bottom-up computation, which can be contrasted with BUC [2] in traditional OLAP, high-level cells are calculated first, before drilling down to low-level cells they cover. In top-down computation, which can be contrasted with Multi-Way [31] in traditional OLAP, we calculate low-level cells first, and then aggregate to high-level cells. Finally, which approach to adopt will depend on various parameters, including the size of the network, data sparsity, the measures to be computed, and the available constraints.

Now consider the bottom-up approach; on the one hand, if a high-level cell c_h does not satisfy an anti-monotone constraint, then we know that no low-level cell c_l covered by c_h would satisfy it, and thus the calculation can be immediately terminated, pruning c_l and its descendants from the cuboid lattice; on the other hand, if a high-level cell c_h already satisfies a monotone constraint, then we no longer need to perform checkings for any low-level cells covered by c_h because they would always satisfy it. As for top-down computations, the roles of anti-monotonicity and monotonicity are reversed accordingly.

It is easy to see that anti-monotone and monotone properties depend on specific analysis of measures and interestingness functions. Here, since we are working with network data, some graph theoretic studies need to be made. Let us examine a few examples.

Claim Suppose maximum flow is the I-OLAP measure to be calculated, regarding its flow's value $|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$, i.e., the highest amount of transportation a network can carry from s to t, constraint $|f| \geq \delta_{|f|}$ is anti-monotone.

Proof Intuitively, the transporting capability of one company must be smaller than that of all companies together, since there are now more available links for the flow to pass. In fact, as we showed in Section 16.4, the flow value of c_h is no smaller than the flow sum of *all* c_l's that are covered by c_h, which is a condition stronger than the normal anti-monotonicity defined between a high-level cell and a *single* low-level cell it covers. ■

Claim The diameter of an information network G is designated as the maximum shortest path length for all pairs of nodes in G. Now, denote diameter as d and let it be the T-OLAP measure we want to calculate, constraint $d \geq \delta_d$ is monotone.

Proof As we perform roll-up in InfoNet T-OLAP and shrink the network, some nodes are merged into one during this process, which can only shorten relevant paths. ■

Because of space limit, we are not going to list more examples here. Certainly, having the conditions on interestingness classified into anti-monotone and monotone will greatly help us in constructing an iceberg graph cube. Here, combined with top-down/bottom-up calculations, Apriori is a simple and effective principle to align search space pruning with particular computation orders. But unfortunately, we cannot use it to push every constraint into the mining process, as there do exist situations that are neither anti-monotone nor monotone. To this extent, we shall further investigate how the other kinds of more complex constraints can be nicely handled in the future.

16.6 Discovery-Driven InfoNetOLAP

Apart from the type of data being dealt with, another important distinction of InfoNetOLAP from the traditional one is the need for flexibility in manipulating

information networks. Thus, in addition to the uniform drilling of traditional OLAP, where all nodes at a given level of abstraction are simultaneously rolled up or drilled down, e.g., from month to quarter, and then to year, InfoNetOLAP may require *selective drilling* for a given node or neighborhood. For instance, in the coauthor network example, a user could be interested in the collaborative relationship between Yahoo! Labs and related individual researchers. Such an analysis may show strong collaborations between AnHai Doan at Wisconsin and people at Yahoo! Labs. So, if all Wisconsin researchers are merged into a single institution in the same way as Yahoo! Labs, it would be hard to discover such a relationship since, collectively, there would be even stronger collaborations between Wisconsin and Berkeley (instead of Yahoo! Labs), which may overshadow AnHai's link to Yahoo! Labs.

Selective drilling, though promising, may generate an exponential number of combinations, which are too costly to compute and explore. To ensure that one can pinpoint to the "real gold" during exploration, *discovery-driven InfoNetOLAP* should be adopted, i.e., rather than searching the complete cube space, one has to be sure that the planned drilling should help the discovery of interesting knowledge.

16.6.1 Discovery-Driven: Guiding Principles

In the following, we first outline a few guiding principles for discovery-driven InfoNetOLAP.

Discovery Driven by *Rule-based Heuristics*. When performing InfoNetOLAP, we usually have some general intuition about where and when the selective drilling should be focused and at what levels/nodes one is most likely to discover something interesting. For instance, when studying research collaborations, it may not be wise to quickly merge prominent researchers into groups, before exploring some interesting patterns for them at first. As an example, if Raghu Ramakrishnan is merged into an even "bigger" entity like Yahoo! Labs, it may blur the individual relationships that can be discovered; rather, we could keep his own identity in the network, on which clearer patterns with finer granularity are observed. Alternatively, for ordinary graduate students, it seems to be more interesting to group them together or have them absorbed by nearby "hub" nodes, because for such individuals, it is not likely that something significant can stand out from the rest. In this sense, a rule like *delay the merge of 'big' or 'distinct' nodes* could be quite simple to work out and follow, as the system only needs to consult attributes of an entity itself (e.g., how many papers a researcher has published) or explore some very local information (e.g., how many persons he/she has collaborated with) for decision making.

Discovery Driven by *Statistical Analysis*. In many cases, it is beneficial to conduct some global (rather than local) preliminary analysis before choosing any drilling operation. Consider a top-down exploratory scenario where the granularity is currently set at the institution level, e.g., both Yahoo! Labs and University of Wisconsin are treated as a whole; now, if an automatic background computation shows that

the collaboration activities between Yahoo! Labs and AnHai Doan are significantly higher than normal, then it is rewarding to drill-down, because the status of AnHai is like an outlier in Wisconsin; otherwise, such drilling may not disclose anything truly interesting. This is similar to discovery-driven OLAP proposed by [23], but in the environment of information networks. As a simple implementation of this idea, we may take the collaboration frequencies between Yahoo! Labs and every person from Wisconsin, and calculate the variance: If the variance is high, one may set up an indicator which may suggest people to click on "U. of Wisc." and expand it to get a refined view, which is just the situation described in Fig. 16.4. Compared to the first guiding principle, there are no longer simple rules that stipulate whether a drill-down or roll-up should be performed; everything depends on a global statistical analysis about the entities in the network, which aims to find out those interesting (or outlying) phenomena that are worthwhile to explore.

Discovery Driven by *Pattern Mining*. Another way for discovery-driven OLAP is fueled by the potential to discover interesting patterns and knowledge on information networks using data mining techniques: If splitting or merging of certain sets of nodes may lead to the discovery of interesting clusters, frequent patterns, classification schemes, and evolution regularities/outliers, then the drilling should be performed, with the results/patterns demonstrated to users. Otherwise, the drilling will not be performed. Notice that, although such pattern discovery can be executed on the fly at the time of user interaction, the discovery process could be too time-consuming with regard to the user's mouse clicking. Therefore, we suggest the pre-computation of some promising drilling paths as intermediate results to speed up interactive knowledge discovery. It is an interesting research issue to determine the things to be computed in order to facilitate such on-line analysis.

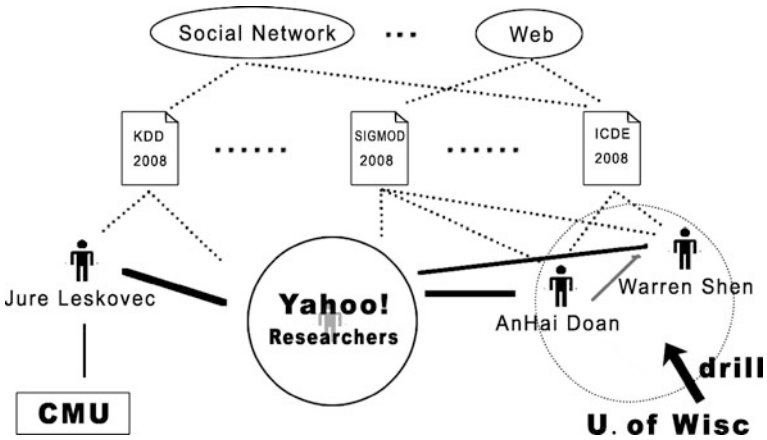


Fig. 16.4 Discovery-driven InfoNetOLAP

16.6.2 Network Discovery for Effective OLAP

As we discussed above, for effective discovery-driven OLAP, it is important to perform some essential data mining on the underlying information networks to reveal interesting network patterns that may help us discover the hidden knowledge. Here, we discuss some interesting network discovery procedures by taking the collaboration scenario of researchers as an example.

For studying coauthor networks, it is important to distinguish different roles the authors may play in the network. For example, based on the coauthor relationships over time, it is possible to dig out advisor–advisee relationships. Usually, advisee is a mediocre node in the network without many publications, who then starts to coauthor substantially with his prominent advisor; after graduation, he/she joins industry or moves on to another institution, and the publishing behaviors are changed again. Advisors can also be identified, based on his/her long-term publication history as well as a center role of working with many junior coauthors. It is interesting to organize researchers under such a phylogeny tree and examine the interactions between different clusters of academic "families".

Besides some not-so-sophisticated knowledge discovery procedures, a mining process may involve induction on the entire information networks as well. For example, in order to partition an interconnected, heterogeneous information network into a set of clusters and rank the nodes in each cluster, one could develop a RankClus framework [26], which integrates clustering and ranking together to effectively cluster information networks into multiple groups and rank nodes in each group based on certain nice properties (such as authority). By examining authors, research papers, and conferences, one can group conferences in the same fields together to form conference clusters, group authors based on their publication records into author clusters, and in the meantime rank authors and conferences based on their corresponding authorities. Such clustering results enhanced by ranking information would be an ideal feed into InfoNetOLAP. Interestingly, such clustering-ranking can be performed based on the links only, without checking the citation information and the keywords or text information contained in the conferences and/or publication titles. The details of such techniques are beyond the discussions of this chapter, but it sheds light on automated processes to effectively identify concept hierarchies and important nodes for discovery-driven InfoNetOLAP.

16.7 Experiments

In this section, we present empirical studies evaluating the effectiveness and efficiency of the proposed InfoNetOLAP framework. It includes two kinds of data sets, one real data set and two synthetic data sets. All experiments are done on a Microsoft Windows XP machine with a 3 GHz Pentium IV CPU and 1GB main memory. Programs are compiled by Visual C++.

16.7.1 Real Data Set

The first data set we use is the DBLP Bibliography (http://www.informatik.uni-trier.de/~ley/db/) downloaded in April 2008. Upon parsing the author field of papers, a coauthor network with multiple snapshots can be constructed, where an edge of weight w is added between two persons if they publish w papers together. We pick a few representative conferences for the following three research areas:

- Database (DB): PODS/SIGMOD/VLDB/ICDE/EDBT
- Data Mining (DM): ICDM/SDM/KDD/PKDD
- Information Retrieval (IR): SIGIR/WWW/CIKM

and also distribute the publications into 5-year bins: (2002, 2007], (1997, 2002], (1992, 1997],.... In this way, we obtain two informational dimensions: *venue* and *time*, on which I-OLAP operations can be performed.

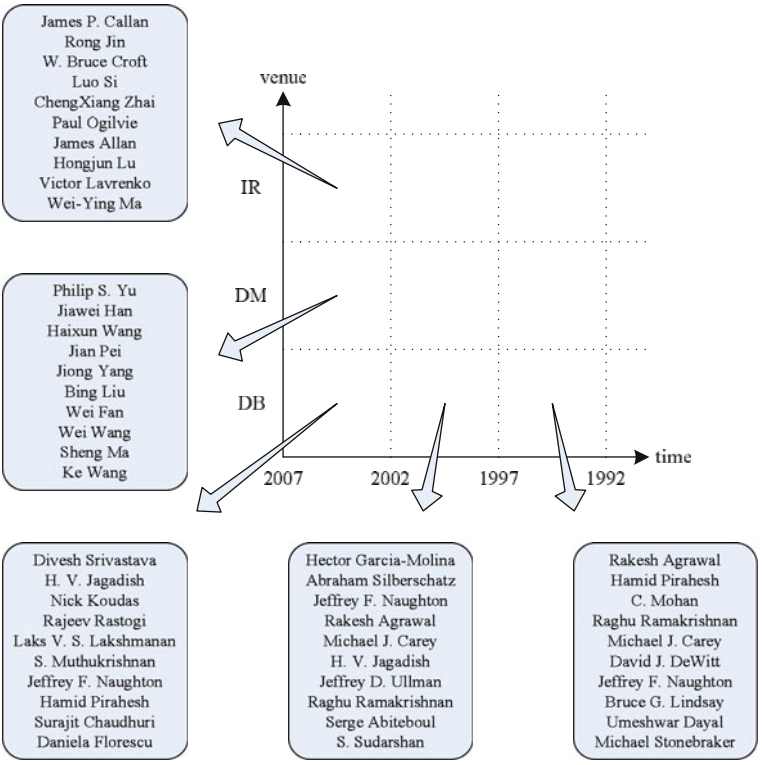


Fig. 16.5 A multi-dimensional view of top-10 "Central" authors

Figure 16.5 shows a classic OLAP scenario. Based on the definition in Section 16.4, we compute the information centrality of each node in the coauthor network and rank them from high to low. In general, people who not only publish a

lot but also publish frequently with a big group of collaborators will be ranked high. Along the *venue* dimension, we can see how the "central" authors change across different research areas, while along the *time* dimension, we can see how the "central" authors evolve over time. In fact, what Fig. 16.5 gives is a multi-dimensional view of the cube's base cuboid; without any difficulty, we can also aggregate DB, DM, and IR into a broad Database field, or generalize the *time* dimension to *all-years*, and then compute respective I-OLAP cells. Given each author's affiliation information, we may conduct T-OLAP and obtain most "central" research groups and institutions as well. The results are omitted here.

16.7.2 Synthetic Data Sets

We use synthetic data sets to demonstrate the effectiveness of the optimizations that are proposed to efficiently perform OLAP operations over information networks. The first test we pick is the computation of maximum flow as an InfoNetI-OLAP measure, which has been used as an exemplifying application in our discussions.

Generator Mechanism. Since it is generally hard to get real flow data, we develop a synthetic generator by ourselves. The data is generated as follows: The network has a source node s and a destination node t, and in between them, there are L intermediate layers, with each layer containing H nodes. There is a link with infinite capacity from s to every node in layer 1, and likewise from every node in layer L to t. Other links are added from layer i to layer $i + 1$ on a random basis: For the total number of $H \cdot H$ choices between two layers, we pick αH^2 pair of nodes and add a link with capacity 1 between them.

For the cube we construct, there are d dimensions; each dimension has $card$ different values (i.e., cardinality), which can be generalized to "all/*". For a base cell where all of its dimensions are set on the finest ungeneralized level, we generate a snapshot of capacity network $L5H1000\alpha 0.01$, i.e., there are five layers of intermediate nodes, and $0.01 \cdot (1000)^2 = 10,000$ links are randomly added between neighboring layers.

The algorithm we use to compute the maximum flow works in an incremental manner. It randomly picks an augmenting path from s to t until no such paths exist. To accommodate top-down computation, where high-level cells are computed after low-level cells so that intermediate results can be utilized, we integrate our attenuation scheme with the classic Multi-Way aggregation method for cube computation [31].

The results are depicted in Figs. 16.6 and 16.7, with Fig. 16.6 fixing the cardinality as 2 and varying d from 2, 3, ..., up to 6, and Fig. 16.7 fixing the number of dimensions as 2 and varying $card$ from 2, 4 ..., up to 8. It can be seen that the optimization achieved through attenuation is obvious, because in effect we do not need to compute a full-scale aggregated graph from scratch, and part of the burden has been transferred to previous rounds of calculations. Especially, when the dimensionality goes high in Fig. 16.6, so that more levels of cube cells are present, the superiority of attenuation-based methods becomes more significant, and one may reap orders of magnitude savings.

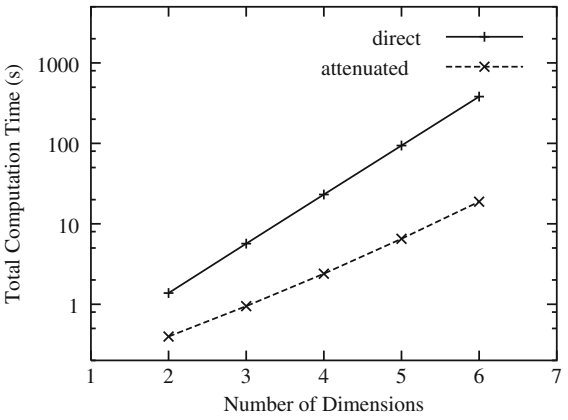


Fig. 16.6 The effect of optimization w.r.t. number of dimensions

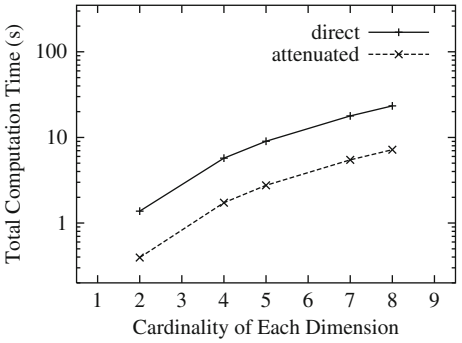


Fig. 16.7 The effect of optimization w.r.t. dimension cardinality

The second test we perform is on the calculation of degree centrality with regard to InfoNet T-OLAP.

Generator Mechanism. The synthetic data networks in this experiment are generated in the following manner: For a network G, n vertices are generated first, i.e., $|V(G)| = n$; as the next step, edges are randomly attached to vertices such that (1) the entire network is connected, (2) the vertices have an average degree of $\overline{d}$, and (3) the edges have an average weight of $\overline{w}$.

Given a network G, users can choose a subset of vertices $V \subseteq V(G)$ to merge into a single vertex v' and compute the T-aggregated graph for the generalized network G'. Such a roll-up operation is called a user T-OLAP request. For a network G, we recursively partition G into π non-overlapping connected components with equal number of vertices, e.g., suppose $|V(G)| = 1024$ and $\pi = 4$, we can first partition G into 4 connected subgraphs with 256 vertices, and then recursively partition these 4 subgraphs until there is only one vertex left. Finally, if we reverse the above sequence of partitioning, a series of T-OLAP requests will be naturally formed, and

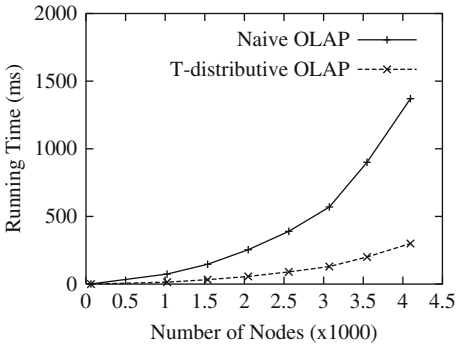


Fig. 16.8 Running time w.r.t. network size

our task next is to compute the corresponding aggregated graphs along the drilling path.

We refer to the baseline algorithm for comparison as Naive OLAP. For each T-OLAP request, this method would directly compute the T-aggregated graph from the original network G. As shown in Section 16.3, degree centrality is T-algebraic: Therefore, we can leverage the (semi-)distributiveness property and make use of the degree centralities that have been calculated for lower-level intermediate networks, which can offer significant efficiency boost. This computation strategy is denoted as T-distributive OLAP.

We set the average vertex degree as $\overline{d} = 5$, and fix the number of partitions per step as $\pi = 4$. Figure 16.8 shows the running time comparison for the two alternative approaches as the number of vertices in G increases. Here, the running time is the total computation cost summed over all T-OLAP requests, and it can be easily observed that with T-distributiveness the computation cost increases much slower than Naive OLAP. In Fig. 16.9, we fix $V(G)$ as 4096 and vary π, which depicts how the granularity of T-OLAP operations can affect both approaches: As π increases, the granularity difference between adjacent levels of networks becomes larger. Since

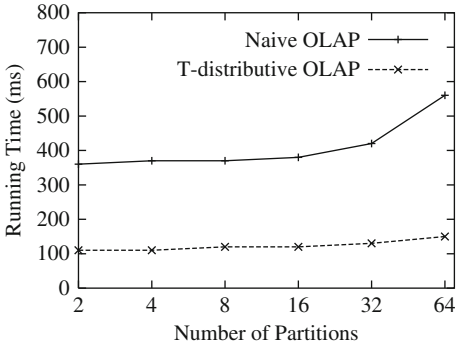


Fig. 16.9 Running time w.r.t. T-OLAP granularity

the computation of degree centrality is not too complex, both approaches have rather slow increase with regard to the running time, as we can see from the picture. However, notice that the T-distributive OLAP still features a flatter growth curve compared with that of the Naive OLAP approach.

16.8 Related Work

OLAP (On-Line Analytical Processing) is an important notion in data mining, which has drawn a lot of attention from the research communities. Representative studies include [7, 11], and a set of papers on materialized views and data warehouse implementations are collected in [12]. There have been a lot of works that deal with the efficient computation of a data cube, such as [2, 31], whereas the wealth of literature cannot be enumerated. However, all these researches target conventional spreadsheet data, i.e., OLAP analysis is performed on independent data tuples that mathematically form a set. In contrast, as far as we know, ours is the first that puts information networks in a rigid multi-dimensional and multi-level framework, where due to the nature of the underlying data, an OLAP measure in general takes the form of an aggregated graph.

The classification of OLAP measures into distributive, algebraic, and holistic was introduced in the traditional OLAP arena, where we can also find related works for iceberg cubing [9], partial materialization [17], and constraint pushing [21]. It is important to see how these basic aspects are dealt with in the InfoNetOLAP scenario; and as we have seen from the discussions, things become much more complicated due to the increased structural complexity.

In InfoNetOLAP, the aggregated graph can be thought as delivering a summarized view of the underlying networks based on some particular perspective and granularity, which helps users get informative insights into the data [6]. In this sense, concerning the generation of summaries for information networks, there have been quite a few researches that are associated with terminologies like compression, summarization, and simplification. For example, Boldi and Vigna [3, 22] study the problem of compressing large graphs, especially Web graphs; however, they only focus on how the Web link information can be efficiently stored and easily manipulated to facilitate computations such as PageRank and authority vectors, which do not provide any pointers into the network structures. Similarly, Beyer and Ramakrishnan [2] develop statistical summaries that analyze simple characteristics like degree distributions and hop-plots on information networks; Lu et al. [18] go one step further by looking into the evolutionary behavior of these statistics and proposes a generative model that helps explain the latent mechanism. These compressed views are useful but hard to be navigated with regard to the underlying networks; also, the multi-dimensional functionality that can conduct analysis from different angles is missing. Another group of papers [1, 15, 19, 29] are often referred as graph simplification, e.g., Archambault et al. [1] aim to condense a large network by preserving its skeleton in terms of topological features, and Kossinets et al. [15] try to extract the "backbone" of a social network, i.e., the subnetwork that consists of edges on

which information has the potential to flow the quickest. In this case, attributes on nodes and edges are not important, and the network is indeed an unlabeled one in its abstract form. Works on graph clustering (to partition similar nodes together), dense subgraph detection (for community discovery, link spam identification, etc.), graph visualization, and evolutionary pattern extraction/contrast include [20], [10], [13, 28], and [5] respectively. They all provide some kind of summaries, but the objective and result achieved are substantially different from those of this chapter.

With regard to summarizing attributed networks that incorporates OLAP-style functionalities, [25] is the closet to ours in spirit. It introduces an operation called SNAP (Summarization by grouping Nodes on Attributes and Pairwise relationships), which merges nodes with identical labels (actually, it might not be necessary to require exactly the same label for real applications, e.g., Lu et al. [18] introduce a way to find similar groups of entities in a network, and this can be taken as the basis to guide node merges), combines corresponding edges, and aggregates a summary graph that displays relationships for such "generalized" node groups. Users can choose different resolutions by a k-SNAP operation just like rolling up and drilling down in an OLAP environment. This can be seen as a special instance of InfoNet T-OLAP that is defined in this chapter.

Sarawagi et al. [23] introduce the discovery-driven concept for traditional OLAP, which aims at pointing out a set of direct handles or indirect paths that might lead to the interesting/outlying cells of a data cube. Their method is statistics oriented. As we proposed in this chapter, InfoNetOLAP can also adopt the discovery-driven concept, but in the context of information networks, which suggests new classes of discovery-driven methods using topological measures and network patterns. Different from traditional OLAP, where dimensions are often globally drilled down and rolled up, InfoNetOLAP takes selective drilling into consideration, which leverages graph mining (e.g., [14]), link analysis (e.g., [24]), etc., and might only involve some local portion of a big network.

16.9 Conclusions

We examine the possibility to apply *multi-dimensional* analysis on information networks, and develop an *InfoNetOLAP framework*, which is classified into two major subcases: *informational OLAP* and *topological OLAP*, based on the different OLAP semantics. Due to the nature of the underlying data, an OLAP measure now takes the form of an *aggregated graph*. We categorize aggregated graphs based on the difficulty to compute them in an OLAP context and suggest two properties: *localization* and *attenuation*, which may help speed up the processing. Both full materialization and constrained partial materialization are discussed. Toward more intelligent InfoNetOLAP, we further propose a *discovery-driven* multi-dimensional analysis model and discuss many challenging research issues associated with it. Experiments show insightful results on real data sets and demonstrate the efficiency of our proposed optimizations.

As for future works, there are a lot of directions we want to pursue on this topic, for example, extending the current framework to heterogeneous-typed information networks, hyper-graphs, etc., and our immediate target would be refining the discovery-driven InfoNetOLAP idea and testing it on several interesting application domains.

References

1. D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, 2007.
2. K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD Conference*, pages 359–370, 1999.
3. P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, pages 595–602, 2004.
2. D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), 2006.
5. J. Chan, J. Bailey, and C. Leckie. Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems*, 16(1):53–96, 2008.
6. V. Chandola and V. Kumar. Summarization – compressing data into an informative representation. *Knowledge Information Systems*, 12(3):355–378, 2007.
7. S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.
8. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.
9. M. Fang, N. Shivakumar, H. Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *VLDB*, pages 299–310, 1998.
10. D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, pages 721–732, 2005.
11. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
12. A. Gupta and I. S. Mumick. *Materialized Views: Techniques, Implementations, and Applications*. MIT Press, Cambridge, MA, 1999.
13. I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.
14. G. Jeh and J. Widom. Mining the space of graph properties. In *KDD*, pages 187–196, 2004.
15. G. Kossinets, J. M. Kleinberg, and D. J. Watts. The structure of information pathways in a social communication network. In *KDD*, pages 435–443, 2008.
18. J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.
17. X. Li, J. Han, and H. Gonzalez. High-dimensional olap: A minimal cubing approach. In *VLDB*, pages 528–539, 2004.
18. W. Lu, J. C. M. J., E. E. Milios, N. Japkowicz, and Y. Zhang. Node similarity in the citation graph. *Knowledge and Information Systems*, 11(1):105–129, 2007.
19. S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD Conference*, pages 419–432, 2008.
20. A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2001.

21. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *SIGMOD Conference*, pages 13–24, 1998.
22. S. Raghavan and H. Garcia-Molina. Representing web graphs. In *ICDE*, pages 405–416, 2003.
23. S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, pages 168–182, 1998.
24. P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *CS-TR-4905, University of Maryland, College Park*, 2008.
25. K. Stephenson and M. Zelen. Rethinking centrality: Methods and examples. *Social Networks*, 11(1):1–37, 1989.
26. Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu. Rankclus: Integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*, pages 565–576, 2009.
25. Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD Conference*, pages 567–580, 2008.
28. N. Wang, S. Parthasarathy, K.-L. Tan, and A. K. H. Tung. Csv: visualizing and mining cohesive subgraphs. In *SIGMOD Conference*, pages 445–458, 2008.
29. A. Y. Wu, M. Garland, and J. Han. Mining scale-free networks using geodesic clustering. In *KDD*, pages 719–724, 2004.
30. X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. F. M. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. S., D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.
31. Y. Zhao, P. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *SIGMOD Conference*, pages 159–170, 1997.

Chapter 17
Integrating Clustering with Ranking in Heterogeneous Information Networks Analysis

Yizhou Sun and Jiawei Han

Abstract Heterogeneous information networks, i.e., the logic networks involving multi-typed, interconnected objects, are ubiquitous. For example, a bibliographic information network contains nodes including authors, conferences, terms and papers, and links corresponding to relations exiting between these objects. Extracting knowledge from information networks has become an important task. Both ranking and clustering can provide overall views on information network data, and each has been a hot topic by itself. However, ranking objects globally without considering which clusters they belong to often leads to dumb results, e.g., ranking database and computer architecture conferences together may not make much sense. Similarly, clustering a huge number of objects (e.g., thousands of authors) into one huge cluster without distinction is dull as well. In contrast, a good cluster can lead to meaningful ranking for objects in that cluster, and ranking distributions for these objects can serve as good features to help clustering. Two ranking-based clustering algorithms, RankClus and NetClus, thus are proposed. RankClus aims at clustering target objects using the attribute objects in the remaining network, while NetClus is able to generate net-clusters containing multiple types of objects following the same schema of the original network. The basic idea of such algorithms is that ranking distributions of objects in each cluster should be quite different from each other, which can be served as features of clusters and new measures of objects can be calculated accordingly. Also, better clustering results can achieve better ranking results. Ranking and clustering can be mutually enhanced, where ranking provides better measure space and clustering provides more reasonable ranking distribution. What's more, clusters obtained in this way are more informative than other methods, given the ranking distribution for objects in each cluster.

Y. Sun (✉)
University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: sun22@illinois.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_17,

17.1 Introduction

In many applications, there exist a large number of individual agents or components interacting with a specific set of components, forming large, interconnected, and sophisticated networks. We call such interconnected networks as *information networks*, with examples including the Internet, highway networks [14], electrical power grids, research collaboration networks [10], public health systems, and biological networks [20]. Clearly, information networks are ubiquitous and form a critical component of modern information infrastructure. Among them, heterogeneous network is a special type of network that contains objects of multiple types. For example, a bibliographic information network extracted from DBLP data[1] contains nodes (including authors, conferences, terms and papers), and links corresponding to relations exiting between these objects. Formally, we can define information network as follows.

Definition 1 (Information Network) Given a set of objects from T types $\mathcal{X} = \{X_t\}_{t=1}^{T}$, where X_t is a set of objects belonging to tth type, a weighted graph $G = \langle V, E, W \rangle$ is called an information network on objects $\mathcal{X}$, if $V = \mathcal{X}$, E is a binary relation on V, and $W : E \to \mathbb{R}^+$ is a weight mapping from an edge $e \in E$ to a real number $w \in \mathbb{R}^+$. Specially, we call such an information network *heterogeneous network* when $T \geq 2$ and *homogeneous network* when $T = 1$.

A great many analytical techniques have been proposed toward a better understanding of information networks, though major on homogeneous information networks, and their properties, among which are two prominent ones: *ranking* and *clustering*. On one hand, *ranking* evaluates objects of information networks based on some *ranking function* that mathematically demonstrates characteristics of objects. With such functions, two objects can be compared, either qualitatively or quantitatively, in a partial order. PageRank [4] and HITS [15], among others, are perhaps the most renowned ranking algorithms over information networks. On the other hand, *clustering* group objects based on a certain proximity measure so that similar objects are in the same cluster, whereas dissimilar ones are in different clusters. After all, as two fundamental analytical tools, ranking and clustering demonstrate overall views of information networks, and hence be widely applied in different information network settings.

Clustering and ranking are often regarded as *orthogonal* techniques, each of which is applied separately to information network analysis. However, applying either of them over information networks often leads to incomplete, or sometimes rather biased, analytical results. For instance, ranking objects over the global information networks without considering which clusters they belong to often leads to dumb results, e.g., ranking database and computer architecture conferences and authors together may not make much sense; alternatively, clustering a large number of objects (e.g., thousands of authors) in one cluster without distinction is dull as

[1] www.informatik.uni-trier.de/~ley/db/

well. However, combining both functions together may lead to more comprehensible results, as shown in Example 1.

Example 1 (Ranking Without/With Clustering) Consider a set of conferences from two areas of DB/DM (i.e., *Database and Data Mining*) and HW/CA (i.e., *Hardware and Computer Architecture*), each having 10 conferences, as shown in Table 17.1. Then we choose 100 authors in each area from DBLP [6]. With the authority ranking function specified in Section 17.2.2, our ranking-only algorithm gives top-10 ranked results (Table 17.2a). Clearly, the results are rather dumb (because of the mixture of the areas) and are biased toward (i.e., ranked higher for) the HW/CA area. What is more, such dull or biased ranking result is caused not by the specific ranking function we chose but by the inherent incomparability between the two areas.

Still consider the same data set, this time we picked 10 conferences in the DB/DM area and rank them as well as the authors relative to this conference cluster. The ranking results are shown in Table 17.2b.

Example 1 shows that good cluster indeed enhances ranking results. Moreover, assigning ranks to objects often leads to better understanding of each cluster. Obviously, good clusters promote good ranking, but how to get good clusters? Before answering this question, we need to first clarify the concept of clusters in heterogeneous networks. In this work, we are going to address two types of clusters, which are solved by two algorithms respectively.

The first type of cluster contains a single type of objects, which are target objects. A straightforward way to generate such clusters in a heterogeneous network is to first evaluate similarity between objects using a link-based method, such as SimRank [13], and then apply graph clustering methods [17, 21] or the like to generate clusters. However, to evaluate similarity between objects in an arbitrary multi-typed information network is a difficult and time-consuming task. Instead,

Table 17.1 A set of conferences from two research areas

DB/DM	{SIGMOD, VLDB, PODS, ICDE, ICDT, KDD, ICDM, CIKM, PAKDD, PKDD}
HW/CA	{ASPLOS, ISCA, DAC, MICRO, ICCAD, HPCA, ISLPED, CODES, DATE, VTS }

Table 17.2 Top-10 ranked conferences and authors without/with clustering

(a) Ranking without clustering				(b) Ranking in DB/DM cluster			
Rank	Conf.	Rank	Authors	Rank	Conf.	Rank	Authors
1	DAC	1	Alberto L. Sangiovanni-Vincentelli	1	VLDB	1	H. V. Jagadish
2	ICCAD	2	Robert K. Brayton	2	SIGMOD	2	Surajit Chaudhuri
3	DATE	3	Massoud Pedram	3	ICDE	3	Divesh Srivastava
4	ISLPED	4	Miodrag Potkonjak	4	PODS	4	Michael Stonebraker
5	VTS	5	Andrew B. Kahng	5	KDD	5	Hector Garcia-Molina
6	CODES	6	Kwang-Ting Cheng	6	CIKM	6	Jeffrey F. Naughton
7	ISCA	7	Lawrence T. Pileggi	7	ICDM	7	David J. DeWitt
8	VLDB	8	David Blaauw	8	PAKDD	8	Jiawei Han
9	SIGMOD	9	Jason Cong	9	ICDT	9	Rakesh Agrawal
10	ICDE	10	D. F. Wong	10	PKDD	10	Raghu Ramakrishnan

RankClus explores rank distribution for each cluster to generate new measures for target objects, and then the clusters are improved under the new measure space. In all, instead of combining ranking and clustering in a two-stage procedure like facet ranking [5, 28], the quality of clustering and ranking can be mutually enhanced in RankClus [23].

The second type of cluster is called net-cluster. In contrast to traditional cluster definition, net-cluster contains multiple types of objects and follows the schema of the original heterogeneous network. A net-cluster can be viewed as a sub-network of original network, with the meaning of a community. To detect net-clusters in arbitrary networks is rather complex; therefore, we only address the problem in a special type of network that with star network schema, in the work of NetClus [24]. A net-cluster example is shown as in Example 2.

Example 2 (Net-Cluster of Database Area) A cluster of the *database area* consists of a set of database authors, conferences, terms, and papers and can be obtained by NetClus on the bibliographic network extracted from DBLP data set. NetClus also presents rank scores for authors, conferences, and terms in its own type. With ranking distribution, users can easily grab the important objects in the area. Table 17.3 shows the top ranked conferences, authors, and terms in the area *database*, generated from a 20-conf. data set (i.e., a "four-area" data set) (see details in Section 17.5) using NetClus.

Table 17.3 Ranking description for net-cluster of database research area

Conference	Rank score	Author	Rank score	Term	Rank score
SIGMOD	0.315	Michael Stonebraker	0.0063	database	0.0529
VLDB	0.306	Surajit Chaudhuri	0.0057	system	0.0322
ICDE	0.194	C. Mohan	0.0053	query	0.0313
PODS	0.109	Michael J. Carey	0.0052	data	0.0251
EDBT	0.046	David J. DeWitt	0.0051	object	0.0138
CIKM	0.019	H. V. Jagadish	0.0043	management	0.0113
…	…	…	…	…	…

Both algorithms are very efficient, which are linear to the links in the network. We will introduce the two algorithms in following sections in detail.

The remainder of the paper is organized as follows. Section 17.2 introduces two ranking functions in heterogeneous networks. The RankClus algorithm and NetClus algorithm are introduced in Sections 17.3 and 17.4 respectively. Section 17.5 is the experiment. A related work introduction is given in Sections 17.6, and 17.7 concludes the discussion.

17.2 Ranking Functions

Ranking function is critical in our ranking-based clustering algorithms, which not only provides ranking distributions for objects to distinguish their importance in a

cluster but also serves as a new feature extraction tool to improve clustering quality. However, current ranking functions are mostly defined on homogeneous networks, such as PageRank and HITS. In this section, we introduce ranking functions on a special type of heterogeneous network, bi-type network. Ranking functions on more complex heterogeneous network are discussed in the end of this section.

Definition 2 (Ranking Distribution and Ranking Function) A ranking distribution $P(\mathbf{X})$ on a type of objects X is a discrete probability distribution, which satisfies $P(X = x) \geq 0$ $(\forall x \in X)$ and $\sum_{x \in X} P(X = x) = 1$. A function $f_X : G \rightarrow P(\mathbf{X})$ defined on an information network G is called a ranking function on type X; if given an information network G, it can output a ranking distribution $P(\mathbf{X})$ on X.

Definition 3 (Bi-type Information Network) Given two types of object sets X and Y, where $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, graph $G = \langle V, E \rangle$ is called a bi-type information network on types X and Y, if $V = X \cup Y$ and $E \subseteq V \times V$.

Let $W_{(m+n)\times(m+n)} = \{w_{o_i o_j}\}$ be the adjacency matrix of links, where $o_i, o_j \in V$, and $w_{o_i o_j}$ equals to the weight of link $\langle o_i, o_j \rangle$, which is the observation number of the link. We also use $G = \langle \{X \cup Y\}, W \rangle$ to denote this bi-type information network. In the following, we use X and Y denoting both the object set and their type name. For convenience, we decompose the link matrix into four blocks: W_{XX}, W_{XY}, W_{YX}, and W_{YY}, each denoting a sub-network of objects between types of the subscripts. W thus can be written as

$$W = \begin{pmatrix} W_{XX} & W_{XY} \\ W_{YX} & W_{YY} \end{pmatrix}.$$

Ranking can give people an overall view of a certain set of objects, which is beneficial for people to grasp the most important information in a short time. More importantly, conditional ranks of attribute types can be served as features for each cluster, and each object in target type can be mapped into a low-dimensional measure space. In this section, we propose two ranking functions that could be used frequently in bi-type network similar to conference–author network. In bibliographic network, consider the bi-type information network composed of conferences and authors. Let X be the type of conference, Y be the type of author, and specify conference as the target type for clustering. According to the publication relationship between conferences and authors, we define the *link matrix* W_{XY} as

$$W_{XY}(i, j) = p_{ij}, \text{ for } i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n,$$

where p_{ij} is the number of papers that author j published in conference i, or equally, the number of papers in conference i that are published by author j. According to the co-author relationship between authors, we define the matrix W_{YY} as

$$W_{YY}(i, j) = a_{ij}, \text{ for } i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n,$$

where a_{ij} is the number of papers that author i and author j co-authored. The link matrix denoting the relationship between authors and conferences W_{YX} is equal to W_{XY}^T, as the relationship between authors and conferences is symmetric, and $W_{XX} = 0$ as there are no direct links between conferences. Based on this conference-author network, we define two ranking functions: *Simple Ranking* and *Authority Ranking*.

17.2.1 Simple Ranking

The simplest ranking of conferences and authors is based on the number of publications, which is proportional to the numbers of papers accepted by a conference or published by an author.

Given the information network $G = \langle \{X \cup Y\}, W \rangle$, simple ranking generates the ranking score of type X and type Y as follows:

$$\begin{cases} \mathbf{r}_X(x) = \dfrac{\sum_{j=1}^{n} W_{XY}(x, j)}{\sum_{i=1}^{m}\sum_{j=1}^{n} W_{XY}(i, j)} \\ \mathbf{r}_Y(y) = \dfrac{\sum_{i=1}^{n} W_{XY}(i, y)}{\sum_{i=1}^{m}\sum_{j=1}^{n} W_{XY}(i, j)}. \end{cases} \tag{17.1}$$

The time complexity of Simple Ranking is $O(|E|)$, where $|E|$ is the number of links.

Obviously, simple ranking is only a normalized weighted degree of each object, which considers every link equally important. In this ranking, authors publishing more papers will have higher ranking score, even these papers are all in junk conferences. In fact, simple ranking evaluates the importance of each object according to its immediate neighborhoods.

17.2.2 Authority Ranking

A more useful ranking we propose here is authority ranking function, which gives an object higher ranking score if it has more authority. Ranking authority merely with publication information seems impossible at first, as citation information could be unavailable or incomplete (such as in the DBLP data, where there is no citation information imported from Citeseer, ACM Digital Library, or Google Scholars). However, two simple empirical rules give us the first clues.

- Rule 1: Highly ranked authors publish *many* papers in highly ranked conferences.
- Rule 2: Highly ranked conferences attract *many* papers from *many* highly ranked authors.

Notice that these empirical rules are domain dependent and are usually given by the domain experts who know both the field and the data set well.[2]

From the above heuristics, we define the ranking score of authors and conferences according to each other as follows.

According to Rule 1, each author's score is determined by the number of papers and their publication forums:

$$\mathbf{r}_Y(j) = \sum_{i=1}^{m} W_{YX}(j,i)\mathbf{r}_X(i). \tag{17.2}$$

When author j publishes more papers, there are more nonzero and high weighted $W_{YX}(j,i)$, and when the author publishes papers in a higher ranked conference i, which means a higher $\mathbf{r}_X(i)$, the score of author j will be higher. At the end of each step, $\mathbf{r}_Y(j)$ is normalized by $\mathbf{r}_Y(j) \leftarrow \frac{\mathbf{r}_Y(j)}{\sum_{j'=1}^{n}\mathbf{r}_Y(j')}$.

According to Rule 2, the score of each conference is determined by the quantity and quality of papers in the conference, which is measured by their authors' ranking scores:

$$\mathbf{r}_X(i) = \sum_{j=1}^{n} W_{XY}(i,j)\mathbf{r}_Y(j). \tag{17.3}$$

When there are more papers appearing in conference i, there are more nonzero and high weighted $W_{XY}(i,j)$; if the papers are published by higher ranked author j, the rank score for j, which is $\mathbf{r}_Y(j)$, is higher, and thus the higher score the conference i will get. The score vector is then normalized by $\mathbf{r}_X(i) \leftarrow \frac{\mathbf{r}_X(i)}{\sum_{i'=1}^{m}\mathbf{r}_X(i')}$.

Notice that the normalization will not change the ranking position of an object, but it gives a relative importance score to each object. The two formulas can be rewritten using the matrix form:

$$\begin{cases} \mathbf{r}_X = \dfrac{W_{XY}\mathbf{r}_Y}{\|W_{XY}\mathbf{r}_Y\|} \\ \mathbf{r}_Y = \dfrac{W_{YX}\mathbf{r}_X}{\|W_{YX}\mathbf{r}_X\|}. \end{cases} \tag{17.4}$$

Theorem 1 *The solution to* $\mathbf{r}_X$ *and* $\mathbf{r}_Y$ *given by the iteration formula is the primary eigenvector of* $W_{XY}W_{YX}$ *and* $W_{YX}W_{XY}$ *respectively.*

[2] For example, a statistician may want to change the rules referring to conferences to journals; whereas a bibliographic database that collects papers from all the bogus conferences may need even more sophisticated rules (extracted from the domain knowledge) to guard the ranking quality.

Proof Combining Eqsuations (17.2) and (17.3), we get $\mathbf{r}_X = \frac{W_{XY}\mathbf{r}_Y}{\|W_{XY}\mathbf{r}_Y\|} = \frac{W_{XY}\frac{W_{YX}\mathbf{r}_X}{\|W_{YX}\mathbf{r}_X\|}}{\|W_{XY}\frac{W_{YX}\mathbf{r}_X}{\|W_{YX}\mathbf{r}_X\|}\|} = \frac{W_{XY}W_{YX}\mathbf{r}_X}{\|W_{XY}W_{YX}\mathbf{r}_X\|}$. Thus, $\mathbf{r}_X$ is the eigenvector of $W_{XY}W_{YX}$. The iterative method is the power method [9] to calculate the eigenvector, which is the primary eigenvector. Similarly, $\mathbf{r}_Y$ is the primary eigenvector of $W_{YX}W_{XY}$.

When considering the co-author information, the scoring function can be further refined by a third rule:

- Rule 3: The rank of an author is enhanced if he or she co-authors with many authors or many highly ranked authors.

Using this new rule, we can revise Eqsuation (17.2) as

$$\mathbf{r}_Y(i) = \alpha \sum_{j=1}^{m} W_{YX}(i, j)\mathbf{r}_X(j) + (1-\alpha)\sum_{j=1}^{n} W_{YY}(i, j)\mathbf{r}_Y(j). \tag{17.5}$$

where parameter $\alpha \in [0, 1]$ determines how much weight to put on each factor based on one's belief.

Similarly, we can prove that $\mathbf{r}_Y$ should be the primary eigenvector of $\alpha W_{YX}W_{XY} + (1-\alpha)W_{YY}$, and $\mathbf{r}_X$ should be the primary eigenvector of $\alpha W_{XY}(I - (1-\alpha)W_{YY})^{-1}W_{YX}$. Since the iterative process is a power method to calculate primary eigenvectors, the ranking score will finally converge.

For authority ranking, the time complexity is $O(t|E|)$, where t is the iteration number and $|E|$ is the number of links in the graph. Notice that, $|E| = O(d|V|) \ll |V|^2$ in a sparse network, where $|V|$ is the number of total objects in the network and d is the average link per each object.

Different from simple ranking, authority ranking gives importance score to each object according to the whole network, rather than the immediate neighborhoods, by the score propagation over the whole network.

17.2.3 Alternative Ranking Functions

Although in this section, we only illustrate two possible ranking functions, the general ranking functions are not confined to these two types. In reality, ranking function is not only related to the link property of an information network but also dependent on the hidden ranking rules used by people in some specific domain. Ranking functions should be combined with link information and user rules in that domain. For example, in many other science fields, journals should be given higher weight when considering an author's rank. Second, although ranking functions in this section are defined on bi-type networks, ranking function on heterogeneous networks with more types of objects can be similarly defined. For example, PopRank [19] is a possible framework to deal with heterogeneous network, which takes into account both the impact within the same type of objects and its relations

with other types of objects. The popularity scores of objects are mutually reinforced through the relations with each other, with different impact factors of different types. Ranking objects in information networks, junk, or spam entities are often ranked higher than deserved. For example, authority ranking can be spammed by some bogus conferences that accept any submit papers due to their huge publication number. Techniques that best use expert knowledge such as TrustRank [11] could be used, which can semi-automatically separate reputable, good objects from spam ones, toward a robust ranking scheme. Personalized PageRank [31] that can utilize expert ranking as query and generate ranking distributions given such knowledge could be another choice to integrate with expert knowledge.

17.3 RankClus

The first clustering task we are solving is to cluster one type of objects (target objects) using other types of objects (attribute objects) and the links in the network. For example, given a bi-typed bibliographic network containing conferences and authors, where links exist between conferences and authors, and between authors and authors, we are interested in clustering conferences into different clusters representing different research communities, using the authors and links in the network. In this section, we introduce RankClus algorithm based on the bi-type network and ranking functions defined in Section 17.2.

17.3.1 Overview

The biggest difficulty in clustering target objects in a network is that the features for those objects are not explicitly given, like in traditional attribute-based data set. The general idea of RankClus is to use ranking distribution-derived features to represent each target object, which are low dimensional. What's more? This feature can be further enhanced during the iterations of the algorithms, since good clustering leads to good ranking, and good ranking gives better ranking-based feature.

Specifically, given the bi-type network $G = \langle \{X \cup Y\}, W \rangle$, suppose that we have a random partition on target type X already, how can we use the conditional ranks to improve the clustering results further? Intuitively, for each conference cluster, which could form a research area, the rank of authors conditional on this area should be very distinct and quite different from the rank of authors in other areas. Therefore, for each cluster X_k, conditional rank of Y, $\mathbf{r}_{Y|X_k}$, can be viewed as a rank distribution of Y, which in fact is a measure for cluster X_k. Then, for each object x in X, the distribution of object y in Y can be viewed as a mixture model over K conditional ranks of Y and thus can be represented as a K-dimensional vector in the new measure space.

We first build the mixture model and use EM algorithm to get the component coefficients for each object in Section 17.3.2, then propose the distance measure

between object and cluster in Section 17.3.3, then summarize the algorithm in Section 17.3.4, and finally give some discussions on extending RANKCLUS to arbitrary information networks in Section 17.3.5.

17.3.2 Mixture Model of Conditional Rank Distribution

Example 3 (Conditional Rank as Cluster Feature) Conditional ranks on different clusters are very different from each other, especially when these clusters are correctly partitioned. Still using the data of the two-research-area example proposed in Section 17.1, we rank 200 authors based on two conference clusters, and the two conditional rank distributions are shown in Fig. 17.1. From the figure, we can clearly see that DB/DM authors rank high relative to DB/DM conferences, while rank extremely low relative to HW/CA conferences. The situation is similar for HW/CA authors.

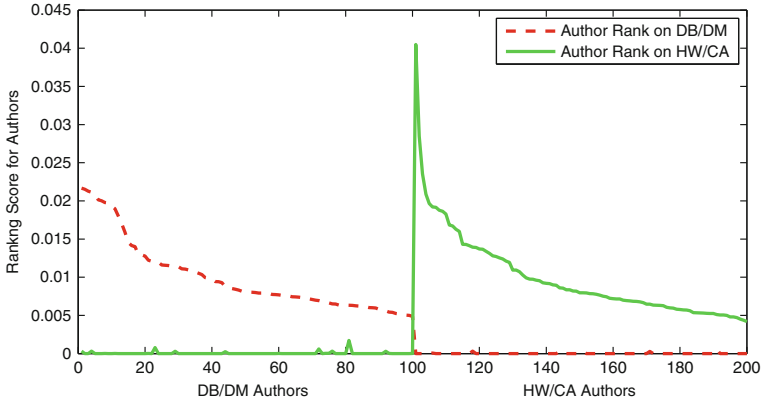


Fig. 17.1 Authors' rank distribution on different clusters

From Example 3, one can see that conditional rank distributions for attribute type on each cluster are quite different from each other and can be used as measures to characterize each cluster. This gives us the intuition to model the distribution for each object x in X over Y as a mixture distribution of K-conditional rank distributions over Y. Here, we only consider the simple case that there are no links between target objects, i.e., $W_{XX} = 0$, and more complex situations will be discussed in Section 17.3.5.

17.3.2.1 Mixture Model for Each Target Object

Suppose we now know the clustering results for type X, which are $X_1, X_2, \ldots,$ and X_K. Also, according to some given ranking function, we

have got conditional rank distribution over Y on each cluster X_k, which is $\mathbf{r}_{Y|X_k}(k = 1, 2, \ldots, K)$, and conditional rank over X, which is $\mathbf{r}_{X|X_k}(k = 1, 2, \ldots, K)$. For simplicity, we use $p_k(Y)$ to denote $\mathbf{r}_{Y|X_k}$ and $p_k(X)$ to denote $\mathbf{r}_{X|X_k}$ in the following deduction. For each object $x_i (i = 1, 2, \ldots, m)$ in X, it follows a distribution $p_{x_i}(Y) = p(Y|x_i)$ to generate a link between x_i and y in Y. Moreover, this distribution could be considered as a mixture model over K-component distributions, which are attribute type's conditional rank distributions on K clusters. We use $\pi_{i,k}$ to denote x_i's coefficient for component k, which in fact is the posterior probability that x_i is generated from cluster k. Thus, $p_{x_i}(Y)$ can be modeled as

$$p_{x_i}(Y) = \sum_{k=1}^{K} \pi_{i,k} p_k(Y) \quad \text{and} \quad \sum_{k=1}^{K} \pi_{i,k} = 1. \tag{17.6}$$

$\pi_{i,k}$ in fact is the probability that object x_i belongs to cluster k, $p(k|x_i)$. Since $p(k|x_i) \propto p(x_i|k)p(k)$, and we have already known $p(x_i|k)$, which is the conditional rank of x_i in cluster k, the goal is thus to estimate the prior of $p(k)$, which is the probability that a link between object x and y belongs to cluster k. In DBLP scenario, a link is a paper, and papers with the same conference and author will be considered as the same papers (since we do not have additional information to discriminate them). The cluster of conference, e.g., DB conferences, can induce a sub-network of conferences and authors with the semantic meaning of DB research area. $p(k)$ is the proportion of papers that belong to the research area induced by the kth conference cluster. Notice that, we can just set the priors as uniform distribution, and then $p(k|x_i) \propto p(x_i|k)$, which means the higher its conditional rank on a cluster, the higher possibility that the object will belong to that cluster. Since conditional rank of X is the propagation score of conditional rank of Y, we can see that highly ranked attribute object has more impact on determining the cluster label of target object.

To evaluate the model, we also make an independence assumption that an attribute object y_j issuing a link is independent to a target object x_i accepting this link, which is $p_k(x_i, y_j) = p_k(x_i)p_k(y_j)$. This assumption says once an author writes a paper, he is more likely to submit it to a highly ranked conference to improve his rank; while for conferences, they are more likely to accept papers coming from highly ranked authors to improve its rank as well.

Example 4 (Component Coefficients as Object Attributes) Following Example 3, each conference x_i is decomposed as a two-dimensional vector $(\pi_{i,1}, \pi_{i,2})$, each dimension stands for the component coefficient. Figure 17.2 is the scatter plot for each conference's two-component coefficients, and different shapes of points represent different areas the conferences really belong to. From the figure, we can see that DB/DM conferences and HW/CA conferences are separated clearly under the new attributes.

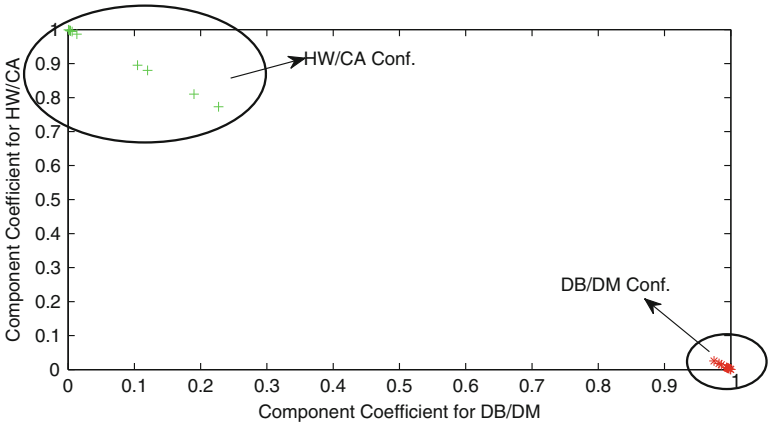


Fig. 17.2 Conferences' scatter plot based on two-component coefficients

17.3.2.2 Parameter Estimation Using EM Algorithm

Next, let's address the problem to estimate the component coefficients in the mixture model. Let Θ be the parameter matrix, which is an $m \times K$ matrix: $\Theta_{m\times K} = \{\pi_{i,k}\}(i = 1, 2, \ldots, m; k = 1, 2, \ldots, K)$. Our task now is to evaluate the best Θ, given the links we observed in the network. For all the links W_{XY} and W_{YY}, we have the likelihood of generating all the links under parameter Θ as

$$\begin{aligned} &L'(\Theta|W_{XY}, W_{YY}) = p(W_{XY}|\Theta)p(W_{YY}|\Theta) \\ =&\prod_{i=1}^{m}\prod_{j=1}^{n} p(x_i, y_j|\Theta)^{W_{XY}(i,j)} \prod_{j=1}^{n}\prod_{j=1}^{n} p(y_i, y_j|\Theta)^{W_{YY}(i,j)}, \end{aligned}$$

where $p(x_i, y_j|\Theta)$ is the probability to generate link $\langle x_i, y_j\rangle$, given current parameter. Since $p(W_{YY}|\Theta)$ does not contain variables from Θ, we only need to consider maximizing the first part of the likelihood to get the best estimation of Θ. Let $L(\Theta|W_{XY})$ be the first part of likelihood. As it is difficult to maximize L directly, we apply EM algorithm [3] to solve the problem.

The initial parameter is Θ^0, which could be set as $\pi^0_{i,k} = \frac{1}{K}$, for all i and k. For conditional distribution $p(z = k|y_j, x_i, \Theta^0)$, it can be calculated using Bayesian rule as follows:

$$p(z = k|y_j, x_i, \Theta^0) \propto p(x_i, y_j|z = k, \Theta^0)p(z = k|\Theta^0) = p^0_k(x_i)p^0_k(y_j)p^0(z = k) \tag{17.7}$$

Thus, integrating into Equation (17.7), we can get the new estimation for $p(z = k)$, given Θ^0 in M-Step of EM algorithm:

$$p(z=k)=\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}W_{XY}(i,j)p(z=k|x_i,y_j,\Theta^0)}{\sum_{i=1}^{m}\sum_{j=1}^{n}W_{XY}(i,j)}. \tag{17.8}$$

Finally, each parameter $\pi_{i,k}$ in Θ is calculated using Bayesian rule:

$$\pi_{i,k}=p(z=k|x_i)=\frac{p_k(x_i)p(z=k)}{\sum_{l=1}^{K}p_l(x_i)p(z=l)}. \tag{17.9}$$

By setting $\Theta^0=\Theta$, the whole process can be repeated. At each iteration, updating rules from Equations (17.7–17.9) are applied, and finally Θ will converge to a local maximum.

17.3.3 Cluster Centers and Distance Measure

After we get the estimations for component efficient for each target object x_i by evaluating mixture models, x_i can be represented as a K-dimensional vector $\mathbf{s}_{x_i}=(\pi_{i,1},\pi_{i,2},\ldots,\pi_{i,K})$. The centers for each cluster can thus be calculated accordingly, which is the mean of $\mathbf{s}_{x_i}$ for all x_i in each cluster. Next, the distance between an object and cluster $D(x,X_k)$ is defined by 1 minus cosine similarity. When initial clusters are randomly partitioned, the initial conditional ranking would be quite similar to each other. In this case, it is possible that all the objects are mixed together and all belong to one cluster in terms of $p_{i,k}$. An example is shown in Fig. 17.3b; conditional rank distributions on Cluster 1 and Cluster 2 are similar to each other, and rank distribution on Cluster 2 dominates Cluster 1 in more data points. As a result, almost every object will have a higher coefficient relative to Cluster 2.

17.3.4 RankClus: Algorithm Summarization

The input for RankClus is the bi-type information network $G=\langle\{X\cup Y\},W\rangle$, the ranking function f, and the cluster number K. The output is K clusters of X with within-cluster rank scores for each x, and conditional rank scores for each y. RankClus is mainly composed of three steps, put in an iterative refinement manner. First, rank for each cluster. Second, estimate the parameter Θ in the mixture model, get new representations $\mathbf{s}_x$ for each target object and $\mathbf{s}_{X_k}$ for each target cluster. Third, adjust each object in type X, calculate the distance from it to each cluster center, and assign it to the nearest cluster:

- Step 0: Initialization. In the initialization step, generate initial clusters for target objects, i.e., assign each target object with a cluster label from 1 to K randomly.
- Step 1: Ranking for each cluster. Based on current clusters, calculate conditional rank for types Y and X and within-cluster rank for type X. In this step, we also

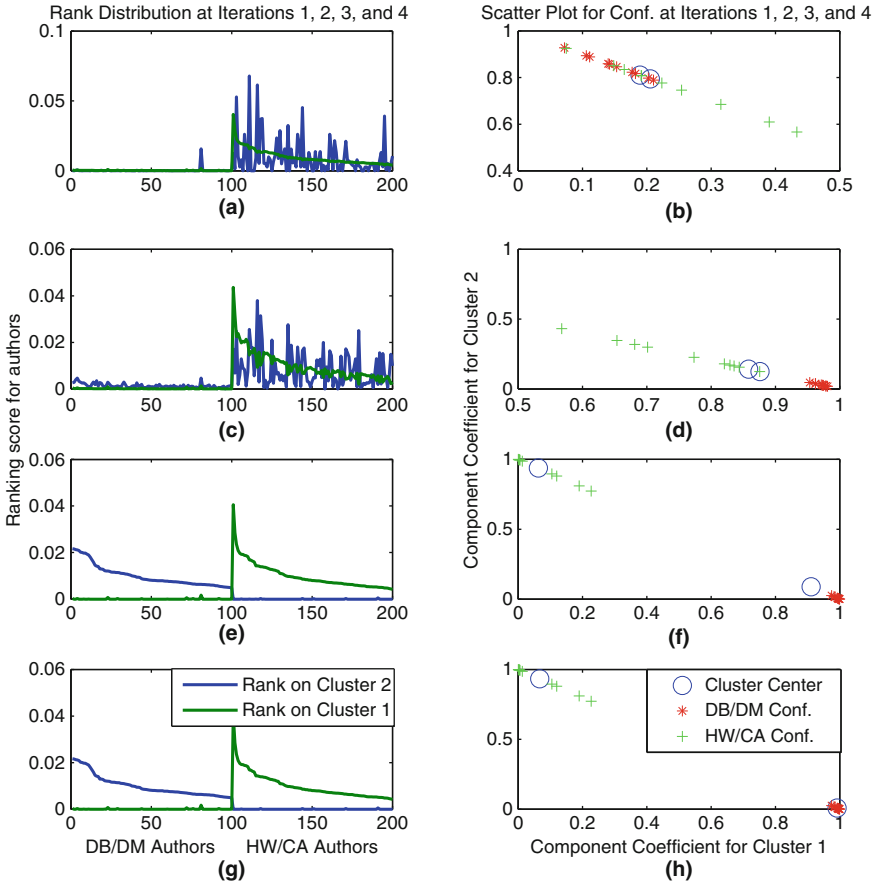


Fig. 17.3 Mutual improvement of clusters and ranking through iterations

need to judge whether any cluster is empty, which may be caused by the improper initialization or biased running results of the algorithm. When some cluster is empty, the algorithm needs to restart in order to generate K clusters.

- Step 2: Estimation of the mixture model component coefficients. Estimate the parameter Θ in the mixture model and get new representations for each target object and centers, for each target cluster: $\mathbf{s}_x$ and $\mathbf{s}_{X_k}$. In practice, the iteration number t for calculating Θ only needs to be set to a small number. Empirically, $t = 5$ can achieve best results.
- Step 3: Cluster adjustment. Calculate the distance from each object to each cluster center and assign it to the nearest cluster.
- Repeat Steps 1, 2, and 3 until clusters change only by a very small ratio ε, or the iteration number should be bigger than a predefined number $iterNum$. In practice, we can set $\varepsilon = 0$ and $iterNum = 20$. Through our experiments, the

algorithm will converge less than 5 rounds in most cases for the synthetic data set and around 10 rounds for DBLP data.

Example 5 (Mutual Improvement of Clustering and Ranking) We now apply our algorithm to the two-research-area example. The conditional rank and component coefficients for each conference at each iteration of the running procedure are illustrated in Fig. 17.3 through (a–h). To better explain how our algorithm can work, we set an extremely bad initial clustering as the initial state. In Cluster 1, there are 14 conferences, half from DB/DM area and half from HW/CA area. Accordingly, Cluster 2 contains the remaining six conferences, which are ICDT, CIKM, PKDD, ASPLOS, ISLPED, and CODES. We can see that the partition is quite unbalanced according to the size and quite mixed according to the area. During the first iteration, the conditional rank distributions for two clusters are very similar to each other (Fig. 17.3a), and conferences are mixed up and biased to Cluster 2 (Fig. 17.3b); however, we can still adjust their cluster label according to the cluster centers, and most HW/CA conferences become the Cluster 2 and most DB/DM conferences become Cluster 1. At the second iteration, conditional ranking is improved a little (shown in Fig. 17.3c) since the clustering (Fig. 17.3b) is enhanced, and this time clustering results (Fig. 17.3d) are enhanced dramatically, although they are still biased to one cluster (Cluster 1). At the third iteration, ranking results are improved dramatically. Clusters and ranks are adjusted afterward, both are minor refinements.

At each iteration, the time complexity of RankClus compriss three parts: ranking part, mixture model estimation part, and clustering adjustment part. For clustering adjustment, we need to compute the distance between each object (m) and each cluster (K), and the dimension of each object is K, so the time complexity for this part is $O(mK^2)$. For ranking, if we use simple ranking, the time complexity is $O(|E|)$. If we use authority ranking, the time complexity is $O(t_1|E|)$, where $|E|$ is the number of links, and t_1 is the iteration number of ranking. For mixture model estimation, at each round, we need to calculate $O(K|E|+K+mK)$ parameters. So, overall, the time complexity is $O(t(t_1|E|+t_2(K|E|+K+mK)+mK^2))$, where t is the iteration number of the whole algorithm and t_2 is the iteration number of the mixture model. If the network is a sparse network, the time is almost linear with the number of objects.

17.3.5 Discussion: Extensions to Arbitrary Multi-typed Information Network

In the previous section, the reasoning of RankClus was based on bi-type networks, with the constraint that there are no links between target objects (i.e., $W_{XX} = 0$). However, RankClus can be applied to other information network as well. In this section, we introduce the basic idea to use RankClus in an arbitrary network: The key is to generate a new set of attributes *from every attribute type* for each object, and the RankClus algorithm proposed in Section 17.3.4 can be used directly.

1. One-type information network. For one-type information network $G = \langle\{X\}, W\rangle$, the problem can be transformed into bi-type network settings $G = \langle\{X \cup Y\}, W\rangle$, where $Y = X$.
2. Bi-type information network with $W_{XX} \neq 0$. For bi-type information network $W_{XX} \neq 0$, the network can be transformed into a three-type network $G = \langle\{X \cup Z \cup Y\}, W\rangle$, where $Z = X$. In this situation, two sets of parameters Θ_Z and Θ_Y can be evaluated separately, by considering links of W_{XZ} and W_{XY} independently. Therefore, for each object x, there should be $2K$ parameters. The first K parameters are its mixture model coefficients over conditional rank distributions of X, while the second K parameters are its mixture model coefficients over conditional rank distributions of Y.
3. Multi-typed information network. For multi-typed information network $G = \langle\{X \cup Y_1 \cup Y_2 \cup \ldots \cup Y_N\}, W\rangle$, the problem can be solved similar to the second case. In this case, we need to evaluate N sets of parameters, by considering conditional ranks from N types: $Y_1, Y_2, \ldots, Y_N$. So, each object can be represented as an NK-dimensional vector.

17.4 NetClus

Among heterogeneous networks, networks with *star network schema* (called star network) such as bibliographic network centered with papers and tagging network (e.g., http://delicious.com) centered with a tagging event are popular and important. In fact, any n-nary relation set such as records in a relational database can be mapped into a star network, with each relation as the center object and all attribute entities linking to it.

Definition 4 (Star Network Schema) An information network $G = \langle V, E, W\rangle$ on $T + 1$ types of objects $\mathcal{X} = \{X_t\}_{t=0}^{T}$ is called star network schema, if $\forall e = \langle x_i, x_j\rangle \in E, x_i \in X_0 \wedge x_j \in X_t (t \neq 0)$, or vise versa. G is then called a *star network*. Type X_0 is called the center type. X_0 is also called the *target type* and $X_t (t \neq 0)$ are called *attribute types*.

Example 6 (Bibliographic Information Network) A bibliographic network consists of rich information about research papers, each *written* by a group of authors, *using* a set of terms, and *published* in a venue (a conference or a journal). Such a bibliographic network is composed of four types of objects: *authors, venues, terms*, and *papers*. Links exist between papers and authors by the relation of "write" and "written by", between papers and terms by the relation of "contain" and "contained in", between papers and venues by the relation of "publish" and "published by". The topological structure of a bibliographic network is shown in the left part of Fig. 17.4, which forms a *star network schema*, where paper is a center type and all other types of objects are linked via papers.

One possible way to cluster a heterogeneous network is to first extract from it a set of homogeneous networks and then apply traditional graph clustering algorithms. However, such an extraction is an information reduction process: some

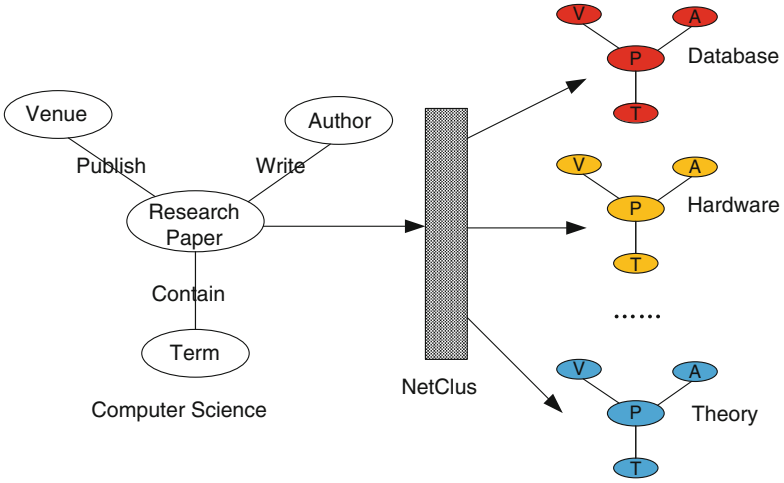


Fig. 17.4 Clustering on a bibliographic network

valuable information, e.g., paper title or venue published in, is lost in an extracted co-author network. NETCLUS is designed on heterogeneous networks with star network schema, which is a ranking-based iterative method following the idea proposed in RankClus [23] that ranking is a good feature to help clustering. However different from RankClus, NETCLUS is able to deal arbitrarily with the number of types of objects, as long as the network is a star network, and also the clusters generated are no more than one type of objects but a network with the same topology as the input network. For a given star network and a specified cluster number K, NETCLUS outputs K net-clusters. Each net-cluster is a sub-layer representing a concept of community of the network, which is an induced network from the clustered target objects, and attached with statistic information for each object in the network. In our algorithm, instead of generating pairwise similarities between objects, which is time consuming and difficult to define under heterogeneous network, NETCLUS maps each target object, i.e., that from the center type, into a K-dimensional vector measure, where K is the cluster number specified by users. The probabilistic generative model for the target objects in each net-cluster is ranking-based, which factorizes a net-cluster into T independent components, where T is the number of attribute types (non-center type).

Further, although clustering co-author network may discover author communities, a research network contains not only authors but also venues, terms, and papers. It is important to preserve such information by directly clustering on heterogeneous networks, which may lead to generating sub-network clusters carrying rich information. This motivates us to develop NETCLUS, a method that discovers *net-clusters*, i.e., a set of sub-network clusters induced from the original heterogeneous network (Fig. 17.4).

Definition 5 (Net-Cluster) Given a network G, a net-cluster C is defined as $C = \langle G', p_C \rangle$, where G' is a *sub-network* of G, i.e., $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$, and $\forall e = \langle x_i, x_j \rangle \in E(G')$, $W(G')_{x_i x_j} = W(G)_{x_i x_j}$. Function $p_C : V(G') \to [0, 1]$ is defined on $V(G')$, for all $x \in V(G')$, $0 \leq p_C(x) \leq 1$, which denotes the probability that x belongs to cluster C, i.e., $P(x \in C)$.

17.4.1 Framework of NetClus Algorithm

Here, we first introduce the general framework of NetClus, and each part of the algorithm will be explained in detail in the following sections. The general idea of the NetClus algorithm given cluster number K is composed of the following steps:

- Step 0: Generate initial partitions for target objects and induce initial net-clusters from the original network according to these partitions, i.e., $\{C_k^0\}_{k=1}^K$.
- Step 1: Build ranking-based probabilistic generative model for each net-cluster, i.e., $\{P(x|C_k^t)\}_{k=1}^K$.
- Step 2: Calculate the posterior probabilities for each target object ($p(C_k^t|x)$) and then adjust their cluster assignment according to the new measure defined by the posterior probabilities to each cluster.
- Step 3: Repeat Steps 1 and 2 until the cluster does not change significantly, i.e., $\{C_k^*\}_{k=1}^K = \{C_k^t\}_{k=1}^K = \{C_k^{t-1}\}_{k=1}^K$.
- Step 4: Calculate the posterior probabilities for each attribute object ($p(C_k^*|x)$) in each net-cluster.

17.4.2 Probabilistic Generative Model for Target Objects in a Net-Cluster

According to many studies in real networks [8, 18], preferential attachment and assortative mixing exist in many real networks, which means an object with a higher degree (i.e., high occurrences) has more probability to be attached with an edge, and higher occurrence objects are more likely to link to each other. As in DBLP data set, 7.64% of the most productive authors publishes 74.2% of all the papers, among which 56.72% papers are published in merely 8.62% of the biggest venues, which means large size conferences and productive authors are intended to co-appear via papers. We extend the heuristic by using ranking, which denotes the overall importance of an object in a network, instead of degree. The intuition is that degree may not represent global importance of an object well. Examples include webpage spammed by many low-rank webpages linking to it (high degree but low rank) will not have too much chance to get a link from a real important webpage, and authors publishing many papers in junk conferences will not increase his/her chance to publish a paper in highly ranked conferences. Under this observation, we simplify the network structure by proposing a probabilistic generative model

for target objects, where a set of highly ranked attribute objects are more likely to co-appear to generate a center object. To explain this idea, we take bibliographic information network as a concrete example and show how the model works. Bibliographic information network as illustrated in Example 5.1 is formalized as follows.

- Bibliographic information network: $G = \langle V, E, W \rangle$.
- Nodes in G: V. In bibliographic network, V is composed of four types of objects: *author* set denoted as A, *conference* set as C, *term* set as T, and *paper* set as D. Suppose the number of distinct objects in each type are $|A|, |C|, |T|$, and $|D|$ respectively, objects in each type are denoted as $A = \{a_1, a_2, \ldots, a_{|A|}\}$, $C = \{c_1, c_2, \ldots, c_{|C|}\}$, $T = \{t_1, t_2, \ldots, t_{|T|}\}$, and $D = \{d_1, d_2, \ldots, d_{|D|}\}$. V is the union of all the objects in all types: $V = A \cup C \cup T \cup D$.
- Edges in G: E and W. In bibliographic network, each paper is written by several authors, published in one conference, and contains several terms in the title. Titles of papers are treated as a bag of terms, in which the order of terms is unimportant but the number of occurrence of terms is. Therefore, for each paper $d_i, i = 1, 2, \ldots, |D|$, it has three kinds of links, going to three types of attribute objects respectively. For two objects from two arbitrary types, x_i and x_j, if there is a link between them, then edge $\langle x_i, x_j \rangle \in E$. Notice that the graph we consider here is an undirected graph. Also, we use $w_{x_i x_j}$ to denote the weight of the link of edge $\langle x_i, x_j \rangle$, which is defined as follows:

$$w_{x_i x_j} = \begin{cases} 1, \text{ if } x_i(x_j) \in A \cup C \text{ and } x_j(x_i) \in D, \\ \quad \text{and } x_i \text{ has link to } x_j \\ c, \text{ if } x_i(x_j) \in T \text{ and } x_j(x_i) \in D \text{ and } x_i(x_j) \text{ .} \\ \quad \text{appears c times in } x_j(x_i), \\ 0, \text{ otherwise.} \end{cases}$$

In order to simplify the complex network with multiple types of objects, we try to factorize the impact of different types of attribute objects and then model the generative behavior of target objects. The idea of factorizing a network is as follows: we assume that given a network G, the probability to visit objects from different attribute types is independent of each other. Still, the probability to visit an attribute object in G, say author a_i, can be decomposed into two parts: $p(a_i|G) = p(A|G) \times p(a_i|A, G)$, where the first part $p(A|G)$ is the overall probability that type of author will be visited in G and the second part $p(a_i|A, G)$ is the probability that an object a_i will be visited among all the authors in the network G. Generally, given an attribute object x and its type T_x, the probability to visit x in G is defined as in Equation (17.10):

$$p(x|G) = p(T_x|G) \times p(x|T_x, G). \tag{17.10}$$

In practice, $p(T_x|G)$ can be estimated by the proportion of objects in T_x compared with the whole attribute object set $\bigcup T_x$ for all attribute types. Later we will show

that the value of $p(T_x|G)$ is not important and can be set to 1. How to generate ranking distribution $p(x|T_x, G)$ for type T_x in a given network G will be addressed in Section 17.4.4.

Also, we make another independence assumption that within the same type of objects, the probability to visit two different objects is independent of each other:

$$p(x_i, x_j|T_x, G) = p(x_i|T_x, G) \times p(x_j|T_x, G),$$

where $x_i, x_j \in T_x$, and T_x are some attribute types.

Now, we build the generative model for target objects, given the ranking distributions of attribute objects in the network G. Still using bibliographic network as an example, each paper d_i is written by several authors, published in one conference, and comprised of a bag of terms in the title. Therefore, a paper d_i is determined by several attribute objects, say $x_{i1}, x_{i2}, \ldots, x_{in_i}$, where n_i is the number of links d_i has. The probability to generate a paper d_i is equivalent to generating these attribute objects with the occurrence number indicated by the weight of the edge. Under the independency assumptions that we have made, the probability to generate a paper d_i in the network G is defined as follows:

$$p(d_i|G) = \prod_{x \in N_G(d_i)} p(x|G)^{W_{d_i,x}} = \prod_{x \in N_G(d_i)} p(x|T_x, G)^{W_{d_i,x}} p(T_x|G)^{W_{d_i,x}},$$

where $N_G(d_i)$ is the neighborhood of object d_i in network G, and T_x is used to denote the type of object x. Intuitively, a paper is generated in a cluster with high probability, if the conference it is published in, authors writing this paper, and terms appeared in the title have high probability in that cluster.

17.4.3 Posterior Probability for Target Objects and Attribute Objects

Once we get the generative model for each net-cluster, we can calculate posterior probabilities for each target object. Now the problem is, suppose that we know the generative probabilities for each target object generated from each cluster $k, k = 1, 2, \ldots, K$, then what is the posterior probability that it is generated from cluster k? Here, K is the cluster number given by user. As some target objects may not belong to any of K net-cluster, we will calculate $K+1$ posterior probabilities for each target object instead of K, where the first K posterior probabilities are calculated for each real existing net-clusters $C_1, C_2, \ldots, C_K$, and the last one in fact is calculated for the original network G. Now, the generative model for target objects in G plays a role as background model, and target objects that are not very related to any clusters will have high posterior probability in background model. In this section, we will introduce the method to calculate posterior probabilities for both target objects and attribute objects.

According to the generative model for target objects, the generative probability for a target object d in the target type D in a sub-network $G_k = G(C_k)$ can be calculated according to the *conditional rankings* of attribute types in that sub-network:

$$p(d|G_k) = \prod_{x \in N_{G_k}(d)} p(x|T_x, G_k)^{W_{d,x}} p(T_x|G_k)^{W_{d,x}}, \tag{17.11}$$

where $N_{G_k}(d)$ denotes for the neighborhood of object d in sub-network G_k. In Equation (17.11), in order to avoid zero probabilities in conditional rankings, each conditional ranking should be smoothed using global ranking with smoothing parameter λ_S, before calculating posterior probabilities for target objects:

$$P_S(X|T_X, G_k) = (1 - \lambda_S) P(X|T_X, G_k) + \lambda_S P(X|T_X, G),$$

where λ_S is a parameter that denotes how much we should utilize the ranking distribution from global ranking.

Smoothing [29] is a well-known technology in information retrieval. One of the reasons that smoothing is required in the language model is to deal with the zero probability problem for missing terms in a document. When calculating generative probabilities of target objects using our ranking-based generative model, we meet a similar problem. For example, for a paper in a given net-cluster, it may link to several objects whose ranking score is zero in that cluster. However, if we simply assign the probability of the target object as zero in that cluster, we cannot use other informative objects to decide which cluster this target object is more likely belonging to. In fact, in initial rounds of clustering, objects may be assigned to wrong clusters, and if we do not use smoothing technique, they may not have the chance to go back to correct clusters (see the case of $\lambda_S = 0$ in Fig. 17.8b).

Once a clustering is given on the input network G, say $C_1, C_2, \ldots, C_K$, we can calculate the probability for each target object (say paper d_i), simply by Bayesian rule: $p(k|d_i) \propto p(d_i|k) \times p(k)$, where $p(d_i|k)$ is the probability that paper d_i is generated from cluster k, and $p(k)$ denotes the relative size of cluster k, i.e., the probability that a paper belongs to cluster k, overall. Here, $k = 1, 2, \ldots, K, K+1$. From this formula, we can see that type probability $p(T|G)$ is just a constant for calculating posterior probabilities for target objects and can be neglected.

In order to get the potential cluster size $p(k)$ for each cluster k, we choose cluster size $p(k)$ that maximizes log-likelihood to generate the whole collection of papers and then use the EM algorithm to get the local optimum for $p(k)$:

$$\log\ L = \sum_{i=1}^{|D|} \log(p(d_i)) = \sum_{i=1}^{|D|} \log \left[\sum_{k=1}^{K+1} p(d_i|k) p(k) \right]. \tag{17.12}$$

We use the EM algorithm to get $p(k)$ by simply using the following two iterative formulas, by initially setting $p^{(0)}(k) = \frac{1}{K+1}$:

$$p^{(t)}(k|d_i) \propto p(d_i|k)p^{(t)}(k);\ \ p^{(t+1)}(k) = \sum_{i=1}^{|D|} p^{(t)}(k|d_i)/|D|.$$

When posterior probability is calculated for each target object in each cluster C_k together with the parent cluster C, where $G(C) = G$, each target object d can be represented as a K-dimensional vector: $\mathbf{v}(d) = (p(1|d), p(2|d), \ldots, p(K|d))$. The center for each cluster C_k can be represented using a K-dimensional vector as well, which is the mean vector of all the target objects belonging to the cluster under the new measure. Next, we calculate cosine similarity between each target object and each center of cluster and assign the target object into the cluster with the nearest center. A new sub-network G_k can be induced by current target objects belonging to cluster k. Following the net-cluster definition (Definition 3), $p_{C_k}(d) = 1$ if object d is assigned to cluster C_k, 0 otherwise. The adjustment is an iterative process, until target objects do not change their cluster label significantly under the current measure. Notice that, when measuring target objects, we do not use the posterior probability for background model. We make such choices with two reasons: first, the absolute value of posterior probability for background model should not affect the similarity between target objects; second, the sum of the first K posterior probabilities reflects the importance of an object in determining the cluster center.

The posterior probabilities for attribute objects $x \in A \cup C \cup T$ can be calculated as follows: $p(k|x) = \sum_{d \in N_G(x)} p(k, d|x) = \sum_{d \in N_G(x)} p(k|d)p(d|x) = \sum_{d \in N_G(x)} p(k|d)\frac{1}{|N_G(x)|}$. It simply says, the probability of a conference belonging to cluster C_k equals to the average posterior probability of papers published in the conference, which is similar for authors. And $p_{C_k}(x)$ in net-cluster definition is set to $p(k|x)$.

Example 7 (A Running Example of Posterior Change) In Table 17.4, we select four objects from four types in the DBLP "four-area" data set to show their posterior probabilities, in four net-clusters and a background model, changing along iterations. Initially, net-clusters are generated from random partitions of papers, each of which is very similar to the original network. Therefore, conditional ranking distributions of each type in each cluster are also very similar to the original ones (background). Thus, posterior probabilities for objects in K initial clusters are similar to each other.[3] However, as similar papers under new measure given by posteriors are grouped together, net-clusters in each area become more and more distinct and objects are gradually assigned with a high posterior probability in the cluster that they should belong to.

[3] Initial absolute posterior prob. to background is sensitive to prior λ_P: the higher λ_P , the larger the value. However, final posterior prob. is not significantly affected by λ_P.

Table 17.4 Illustration of posterior change during iterations for different types of objects

	Conf: KDD					Author: Michael Stonebraker					Term: Relational					Paper: SimRank [13]				
Iter	DB	DM	ML	IR	BG	DB	DM	ML	IR	BG	DB	DM	ML	IR	BG	DB	DM	ML	IR	BG
0	0.21	0.25	0.19	0.18	0.17	0.32	0.20	0.11	0.20	0.17	0.28	0.20	0.17	0.18	0.17	0.01	0.00	0.87	0.00	0.12
1	0.09	0.32	0.12	0.13	0.34	0.35	0.03	0.01	0.21	0.39	0.31	0.12	0.09	0.15	0.34	0.02	0.00	0.48	0.00	0.50
2	0.02	0.37	0.07	0.10	0.44	0.46	0.00	0.00	0.30	0.24	0.34	0.10	0.10	0.17	0.29	0.01	0.01	0.02	0.00	0.96
5	0.02	0.62	0.07	0.16	0.14	0.74	0.00	0.00	0.19	0.07	0.55	0.13	0.12	0.14	0.06	0.00	0.00	0.00	0.92	0.08
10	0.01	0.89	0.02	0.03	0.04	0.95	0.00	0.00	0.01	0.04	0.68	0.15	0.12	0.02	0.03	0.00	0.32	0.00	0.43	0.25
end	0.01	0.92	0.01	0.03	0.03	0.95	0.00	0.00	0.01	0.03	0.68	0.16	0.11	0.02	0.02	0.00	0.99	0.00	0.00	0.01

17.4.4 Ranking Distribution for Attribute Objects

We have introduced ranking functions in Section 17.2, and now clarify the ranking function for the bibliographic network with star network schema, and also give some properties of the two ranking functions for a simple three-typed star network.

17.4.4.1 Simple Ranking

Simple ranking is namely the simple occurrence counting for each object normalized in its own type. Given a network G, ranking distribution for each attribute type of objects is defined as follows:

$$p(x|T_x, G) = \frac{\sum_{y \in N_G(x)} W_{xy}}{\sum_{x' \in T_x} \sum_{y \in N_G(x')} W_{x'y}}, \tag{17.13}$$

where x is an object from type T_x. For example, in bibliographic network, the rank score for a conference using simple ranking will be proportional to the number of its accepted papers.

17.4.4.2 Authority Ranking

Authority ranking for each object is a ranking function that considers the authority propagation of objects in the network, thus will represent more of the visibility over the whole network. For a general star network G, the propagation of authority score from Type X to Type Y through the center type Z is defined as

$$P(Y|T_Y, G) = W_{YZ} W_{ZX} P(X|T_X, G), \tag{17.14}$$

where W_{TZ} and W_{ZX} are the weight matrices between the two types of objects as indexed and can be normalized when necessary. Generally, authority score of one type of objects could be a combination of scores from different types of objects, e.g., that proposed in [19]. It turns out that the iteration method for calculating ranking distribution is the power method to calculate the primary eigenvector of a square matrix denoting the strength between pairs of objects in that certain type, which can be achieved by selecting a walking path (or a combination of multiple paths) in the network.

In the DBLP data set, according to the rules (1) highly ranked conferences accept many good papers published by many highly ranked authors and (2) highly ranked authors publish many good papers in highly ranked conferences, we determine the iteration equation as

$$\begin{aligned} P(C|T_C, G) &= W_{CD} D_{DA}^{-1} W_{DA} P(A|T_A G) \\ P(A|T_A, G) &= W_{AD} D_{DC}^{-1} W_{DC} P(C|T_C, G), \end{aligned} \tag{17.15}$$

where D_{DA} and D_{DC} are the diagonal matrices with the diagonal value equaling to row sum of W_{DA} and W_{DC}. Since all these matrices are sparse, in practice, the rank scores of objects need only be calculated iteratively according to their limited neighbors.

In both ranking functions, prior distributions for a certain type in different clusters can be integrated. Priors for a given type X are given in the form $P_P(X|T_X, k), k = 1, 2, \ldots, K$. A user may give only a few representative objects to serve as priors, like terms and conferences in bibliographic data. First, the prior is propagated in the network in a Personalized PageRank [31] way, to propagate scores to objects that are not given in the priors. Then, the propagated prior is linear combined with the ranking functions with parameter $\lambda_P \in [0, 1]$: The bigger the value, the more the final conditional ranking is dependent on prior.

17.4.5 Algorithm Summary and Time Complexity Analysis

Time complexity of NETCLUS is composed of the following parts. First, computational complexity for global ranking for attribute objects is $O(t_1|E|)$ and that for global probability calculation for target objects is $O(|E|)$, where $|E|$ is the number of edges in network G and t_1 is the iteration number for ranking. For ranking, at each iteration, each link will be calculated once; and for global probability calculation, a link is still calculated once. Second, time complexity for conditional ranking for attribute objects is $O(t_1|E_k|)$, and for conditional probability for target objects is $O(|E_k|)$ in each cluster k. When adding them together, for all sub-clusters, time complexity for one iteration of clustering should be $O(t_1|E|+|E|)$. Third, time complexity for calculating posterior probability for each target object is $O(t_2(K+1)N)$, where N is the number of target objects, and t_2 is the max iteration number in the EM algorithm. Fourth, cluster adjustment for each target object is $O(K^2N)$. Since for each target object, it has a K-dimensional measure, and we have to calculate similarity to K clusters' centers, which are also K-d. Fifth, time complexity for posterior probability for each attribute object is $O(K|E|)$. For each attribute object, each link to target object should be used once to calculate the posterior probability for it. Also, for each attribute type, we have to calculate a K-d measure.

In all, the time complexity for NETCLUS is $O((t_1 + 1)|E| + t_3((t_1 + 1)|E| + t_2(K + 1)N + K^2N) + K|E|)$, where t_3 is max iteration number used for clustering adjustment, which can be summarized as $O(c_1|E| + c_2N)$. When the network is very sparse, which is a real situation in most applications, the time complexity is almost linear to the objects in the network.

17.5 Experiments

In this section, we study the effectiveness and efficiency of RANKCLUS and NETCLUS respectively.

17.5.1 RankClus

First, we will show the effectiveness and efficiency of RANKCLUS algorithm compared with other linked based algorithms, based on both synthetic and real data sets.

17.5.1.1 Accuracy and Efficiency Study on Synthetic Data

In order to compare accuracy among different clustering algorithms, we generate synthetic bi-type information networks, which follow the properties of real information networks similar to DBLP. Please refer to [23] to see the detail of syntectic data.

In order to evaluate the accuracy of the clustering results, we adopt Normalized Mutual Information measure. For N objects, K clusters, and two clustering results, let $n(i, j), i, j = 1, 2, \ldots, K$, the number of objects that has the cluster label i in the first cluster and the cluster label j in the second cluster. From $n(i, j)$, we can define joint distribution $p(i, j) = \frac{n(i,j)}{N}$, row distribution $p_1(j) = \sum_{i=1}^{K} p(i, j)$, and column distribution $p_2(i) = \sum_{j=1}^{K} p(i, j)$. NMI is defined as $\frac{\sum_{i=1}^{K}\sum_{j=1}^{K} p(i,j)\log(\frac{p(i,j)}{p_1(j)p_2(i)})}{\sqrt{\sum_{j=1}^{K} p_1(j)\log p_1(j)\sum_{i=1}^{K} p_2(i)\log p_2(i)}}$.

We compared RANKCLUS implemented with two ranking functions, which are simple ranking and authority ranking, with state-of-the-art spectral clustering algorithm, which is the k-way Ncut algorithm proposed in [21], implemented with two similarity matrix generation methods, which are Jaccard coefficient and SimRank [13]. Results for accuracy is in Fig. 17.5. For each network configuration, we generate 10 different data sets and run each algorithm 100 times. From the results, we can see that, two versions of RANKCLUS outperform in the first four data sets. RANKCLUS with authority ranking function is even better, since authority ranking gives a better rank distribution, as it is able to utilize the information of the whole network. Through the experiments, we observe that performance of two versions of RankClus and the NCut algorithm based on Jaccard coefficient are highly dependent on the data quality, in terms of cluster separateness and link density. SimRank

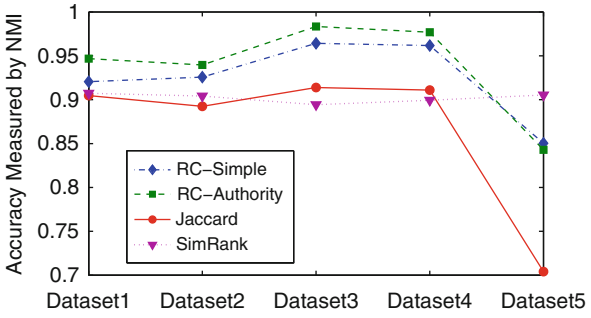


Fig. 17.5 Accuracy of clustering

has a very stable performance. Further experiments show that the performance of SimRank will deteriorate when the data quality is rather poor (when average link for each target object is 40, the NMI accuracy becomes as low as 0.6846).

In order to check the scalability of each algorithm, we set four different size networks, in which both the object size and the link size are increasing by a factor of 2. The average time used by each algorithm for each data set is summarized in Fig. 17.6. We can see that compared with the time-consuming SimRank algorithm, RANKCLUS is also very efficient and scalable.

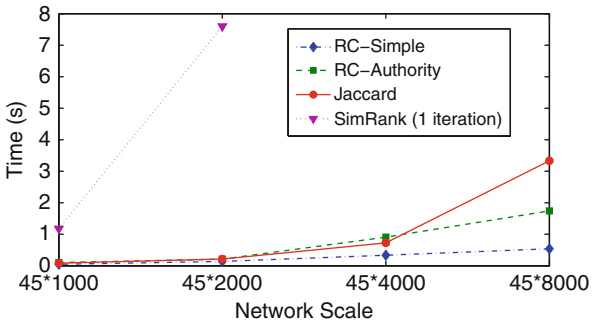


Fig. 17.6 Efficiency analysis

17.5.1.2 Case Study on the DBLP Data Set

We use the DBLP data set to generate a bi-type information network for all the 2676 conferences and 20,000 authors with most publications, from the time period of year 1998 to year 2007. Both conference–author relationships and co-author relationships are used. We set cluster number $K = 15$ and apply RANKCLUS with authority function proposed in Section 17.4.2, with $\alpha = 0.95$. We then pick five clusters and show top-10 conferences from each cluster according to within-cluster scores. For clarify, each research area labels are added manually to each cluster. The results are shown in Table 17.5.

Table 17.5 Top-10 conferences in five clusters using RANKCLUS

	DB	Network	AI	Theory	IR
1	VLDB	INFOCOM	AAMAS	SODA	SIGIR
2	ICDE	SIGMETRICS	IJCAI	STOC	ACM Multimedia
3	SIGMOD	ICNP	AAAI	FOCS	CIKM
4	KDD	SIGCOMM	Agents	ICALP	TREC
5	ICDM	MOBICOM	AAAI/IAAI	CCC	JCDL
6	EDBT	ICDCS	ECAI	SPAA	CLEF
7	DASFAA	NETWORKING	RoboCup	PODC	WWW
8	PODS	MobiHoc	IAT	CRYPTO	ECDL
9	SSDBM	ISCC	ICMAS	APPROX-RANDOM	ECIR
10	SDM	SenSys	CP	EUROCRYPT	CIVR

Please note that the clustering and ranking of conferences and authors shown in Tables 17.5 and 17.1b have not used any keyword nor citation information, the information popularly used in most bibliographic data clustering or ranking systems. It is well recognized that citation information is crucial at judging the influence and impact of a conference or an author in a field. However, by exploring the publication entries only in the DBLP data, the RankClus algorithm can achieve comparable performance as citation studies for clustering and ranking conferences and authors. This implies that the collection of publication entries without referring to the keyword and citation information can still tell a lot about the status of conferences and authors in a scientific field.

17.5.2 NetClus

We now study the effectiveness of NetClus and compare it with state-of-the-art algorithms.

17.5.2.1 Data Set

We use real data set from DBLP and build bibliographic networks according to Example 6. Two networks with different scales will be studied. First, a data set ("all-area" data set) covering all the conferences, authors, papers, and terms from DBLP will be used. Second, we also extract a small data set ("four-area" data set) which contains four areas that are most related to data mining, which are database, data mining, information retrieval, and machine learning. Five representative conferences for each area are picked, and all authors have ever published papers on any of the 20 conferences, all these papers and terms appeared in these titles are included in the network. By using the smaller data set, we want to compare the clustering accuracy with several other methods. Also, parameter study and ranking function study will be carried on based on the "four-area" data set.

17.5.2.2 Case Study

We first show the ranking distributions in net-clusters we discovered using the "all-area" data set, which is generated by using authority ranking for conferences and authors, setting conference type as priors, and setting the cluster number as 8. We show three net-clusters in Table 17.6. Also, we can recursively apply NetClus to sub-networks derived from clusters and discover finer net-clusters. Top-5 authors in a finer net-cluster about XML area, which is derived from database sub-network, are shown in Table 17.7.

17.5.2.3 Study on Ranking Functions

In Section 17.4.4, we proposed two ranking functions, namely simple ranking and authority ranking. Here, we study how low-dimensional measure derived from

Table 17.6 Top-5 conferences in 3 net-clusters

Rank	DB and IS	Theory	AI
1	SIGMOD	STOC	AAAI
2	VLDB	FOCS	UAI
3	ICDE	SIAM J. Comput.	IJCAI
4	SIGIR	SODA	Artif. Intell.
5	KDD	J. Comput. Syst. Sci.	NIPS

Table 17.7 Top-5 authors in "XML" net-cluster

Rank	Author
1	Serge Abiteboul
2	Victor Vianu
3	Jerome Simeon
4	Michael J. Carey
5	Sophie Cluet

ranking distributions improves clustering and how clustering can improve this new measure in turn (Fig. 17.7). Term is fixed to use simple ranking, and conference and author are set to use either authority ranking or simple ranking as two different settings.

First, in order to measure how dissimilar conditional ranking distributions are among different clusters, we calculate average KL divergence, which is denoted as $avg_{KL}(X)$, between each conditional ranking and global ranking for each attribute type X: $avg_{KL}(X) = \frac{1}{K}\sum_{k=1}^{K} D_{KL}(P(X|T_X, G_k)||P(X|T_X, G))$.

Second, in order to measure the goodness of measure generated in each round of clustering, we use the compactness, C_f, of target objects under each round of clustering for ranking function f, which is defined as the average ratio between within-cluster similarity and between-cluster similarity using the new measure: $C_f = \frac{1}{|D|}\sum_{k=1}^{K}\sum_{i=1}^{|D_k|}\frac{s(d_{ki},c_k)}{\sum_{k'\neq k} s(d_{ki},c_{k'})/(K-1)}$.

Third, we trace the accuracy of clustering results for target objects in each round of iteration, which is defined as accuracy $= \frac{1}{|D|}\sum_{i=1}^{D} P_{\text{true}}(\cdot|d_i) \cdot P(\cdot|d_i)$. However, since $|D|$ is very large even in four-area data set, we manually randomly label 100 papers into four clusters and use this paper set to calculate the accuracy.

Fourth, at each iteration of clustering, we calculate the posterior probability for each paper by maximizing the log-likelihood of the whole collection. Here, we also trace the log-likelihood log L along with the clustering iterations, which is defined in Equation (17.12). From Fig. 17.7, we can see authority ranking is better in every measure than simple ranking.

17.5.2.4 Study on Parameters

In our algorithm, there are two parameters: prior parameter (λ_P) and smoothing parameter setting (λ_S). We use clustering accuracy for sampled papers to test the impact of different settings of parameters to the algorithm. By fixing one of them,

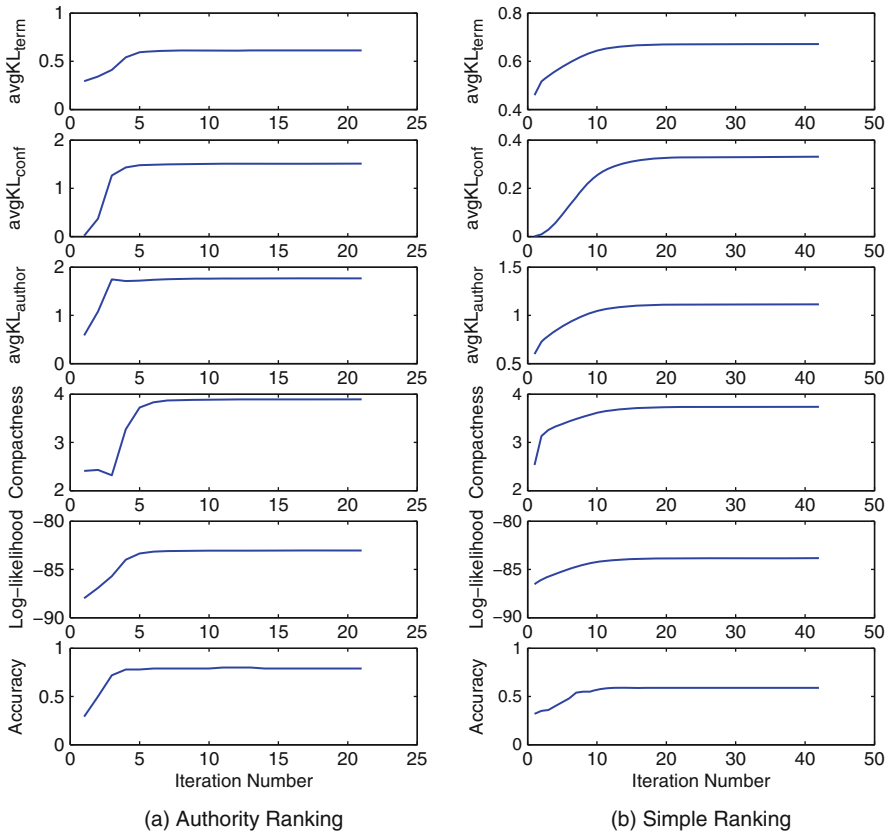


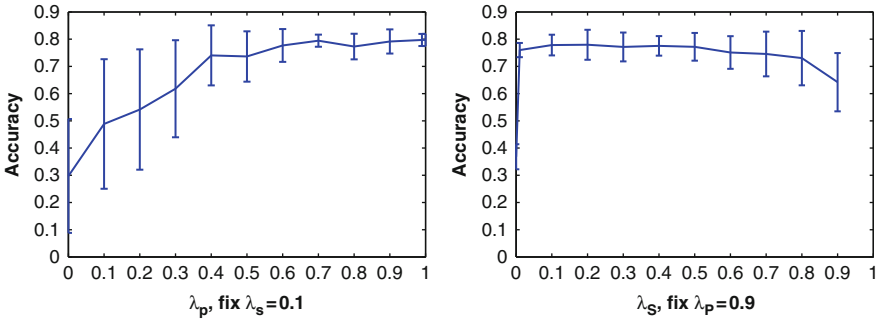
Fig. 17.7 The change of goodness of ranking and clustering along with the iteration number

we vary the other one. From Fig. 17.8a and b, we find that the larger the prior parameter λ_P, the better the results, while when $\lambda_P > 0.4$, the impact becomes more stable[4]; also, the impact of smoothing parameter is very stable, unless it is not too small (less than 0.1) or too big (bigger than 0.8). The results are based on 20 runnings. Priors given for each of the four areas are around two or three terms. For example, "database" and "system" are priors for database area, with uniform prior distribution.

17.5.2.5 Accuracy Study

In this section, we compare our algorithm with two other algorithms. Since all of them cannot be directly applied to heterogeneous network clustering with four types

[4] Actually, the extremely poor quality when λ_P is very small is partially caused by the improper accuracy measure at those occasions. When the prior is not big enough to attract the papers from the correct cluster, the clusters generated not necessarily have the same cluster label with the priors.



(a) Prior: λ_P (b) Smoothing: λ_S

Fig. 17.8 Parameter study of λ_P and λ_S

of objects, for each algorithm, we will simplify the network when necessary to make all the algorithms comparable. For PLSA [30], only the term type and paper type in the network are used. Notice that we use the same term prior in both NETCLUS and PLSA. The accuracy results for papers are in Table 17.8.

Table 17.8 Accu. of paper clustering results

	NetClus $(A + C + T + D)$	PLSA $(T + D)$
Accuracy	**0.7705**	0.608

Since RankClus can only cluster conferences, we choose to measure the accuracy of conference cluster. For NETCLUS, cluster label is obtained according to the largest posterior probability, and NMI [23] is used to measure the accuracy. The results are shown in Table 17.9, where $d(a) > n$ means we select authors that have more than n publications. Since majority authors only publish a few papers, which contains little information for disclosure of the relationship between two conferences and misleads the algorithm, we run RankClus algorithm by setting different thresholds to select subsets of authors. All the results are based on 20 runnings.

Table 17.9 Accu. of conf. clustering results

	RankClus $d(a) > 0$	RankClus $d(a) > 5$	RankClus $d(a) > 10$	NetClus $d(a) > 0$
NMI	0.5232	0.8390	0.7573	**0.9753**

17.6 Related Work

In information network analysis, two most important ranking algorithms are PageRank [4] and HITS [15], both of which are successfully applied to the Internet search. PageRank is a link analysis algorithm that assigns a numerical weight to

each object of the information network, with the purpose of "measuring" its relative importance within the object set. On the other hand, HITS ranks objects are based on two scores: *authority* and *hub*. Authority estimates the value of the content of the object, whereas hub measures the value of its links to other objects. Both PageRank and HITS evaluate the static quality of objects in information network, which is similar to the intrinsic meaning of our ranking methods. However, both PageRank and HITS are designed on the network of webpages, which is a directed homogeneous network, and the weight of the edge is binary. PopRank [19] aims at ranking popularity of web objects. They have considered the role difference of different webpages, and thus turn webpages into a heterogeneous network. They trained the propagation factor between different types of objects according to partial ranks given by experts. Different from their setting, we will calculate the rank for each type of objects separately (i.e., we do not compare ranks of two objects belonging to different types), rather than consider them in a unified framework. J. E. Hirsch [12] proposed h-index originally in the area of physics for characterizing the scientific output of a researcher, which is defined as the number of papers with citation number higher or equal to h. Extended work [22] shows that it can also work well in computer science area. However, h-index will assign an integer value h to papers, authors, and publication forums, while our work requires that rank sores can be viewed as a rank distribution and thus can serve as a good measure for clustering. What is more, since there are only very limited citation information in DBLP, ranking methods demanding citation cannot work in such kind of data. Instead of proposing a totally new strategy for ranking, we aim at finding empirical rules in the specific area of DBLP data set and providing ranking function based on these rules, which works well for the specific case. The real novelty that lies in our framework is that it tightly integrates ranking and clustering and thus offers informative summary for heterogeneous network such as the DBLP data.

Clustering on graphs, often called graph partition, aims at partitioning a given graph into a set of subgraphs based on different criteria, such as minimum cut, min-max cut [7], and normalized cut [21]. Spectral clustering [13, 25] provides an efficient method to get graph partitions which is in fact an NP-hard problem. Rather than investigating the global structure like spectral clustering, several density-based methods [26, 27] are proposed to find clusters in networks which utilize some neighborhood information for each object. These methods are all based on the assumption that the network is homogeneous and the adjacent matrix of the network is already defined. SimRank [13] is able to calculate pairwise similarity between objects by the links of a given network, which could deal with heterogeneous network, such as bipartite network. However, when the structure of network becomes more complex, such as network with star network schema, SimRank cannot give reasonable similarity measures between objects any more. Also, high-time complexity is another issue of SimRank, which prevents it from being applied to large-scale networks. Without calculating the pairwise similarity between two objects of the same type, RankClus and NetClus use conditional ranking as the measure of clusters, and it only needs to calculate the distances between each object and the cluster

center. Recently, a different view of clustering on heterogeneous networks [1, 2, 16] appears, which aims at clustering objects from different types simultaneously. Given different cluster number needed for each type of objects, *clusters for each type* are generated by maximizing some objective function. In NETCLUS, net-cluster follows the original network topology and resembles a community that is comprised of *multiple types* of objects.

In web search, there exists an idea of facet ranking [5, 28], which clusters the returned results for each query into different categories, to help users to better retrieve the relevant documents. A commercial website that illustrates the idea is "vivisimo.com." [5] It may seem that facet ranking also integrates ranking with clustering; however, our work is of totally different idea. First, the goal of facet ranking is to help user to better organize the results. The meaning of ranking here is the relevance to the query. RANKCLUS and NETCLUS aim at finding higher quality and more informative clusters for target objects with rank information integrated in an information network. Second, facet ranking is a two-stage methodology. In the first stage, relevant results are collected according to the relevance of the query, and then clustering is applied on the collection of returned documents. RANKCLUS and NETCLUS integrate ranking and clustering tightly, which are mutually improved during the iterations.

17.7 Conclusions

Ranking-based clustering methods RANKCLUS and NETCLUS [23, 24] are newly proposed algorithms on heterogeneous network analysis. RANKCLUS aims at clustering target objects using the attribute objects in the remaining network, while NETCLUS is able to generate net-clusters containing multiple types of objects following the same schema of the original network. The basic idea of such algorithms is that ranking distributions of objects in each cluster should be quite different from each other. Also, there should be a clustering for target objects, such that every target object is closest to its current cluster given the new measure defined by conditional ranking distributions for each cluster derived from extracted sub-networks. Ranking and clustering can be mutually enhanced, while ranking provides better measure space and clustering provides more reasonable ranking distribution. Clusters generated using such methods are more informative, given the ranking distributions for the objects in the cluster. There are still many research issues to be explored in the framework. Clearly, more research is needed to further consolidate this interesting framework and explore its broad applications.

[5] http://vivisimo.com

References

1. A. Banerjee, S. Basu, and S. Merugu. Multi-way clustering on relation graphs. In *SIAM'07*, pages 145–156, Minneapolis, Minnesota, April 2007.
2. R. Bekkerman, R. El-Yaniv, and A. McCallum. Multi-way distributional clustering via pairwise interactions. In *ICML'05*, pages 41–48, Bonn, Germany, August 2005.
3. J. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models, TR-97-021, April, 1998. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.576. 1997.
4. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
5. D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *SIGIR'92*, pages 318–329, Copenhagen, Denmark, June 1992.
6. DBLP. The dblp computer science bibliography. http://www.informatik.uni-trier.de/ ley/db/.
7. C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *ICDM'01*, pages 107–114. IEEE Computer Society, San Jose, California, USA, November-December 2001.
8. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM'99*, pages 251–262, Cambridge, Massachusetts, USA, August-September 1999.
9. J. E. Gentle and W. HSrdle. *Handbook of Computational Statistics: Concepts and Methods*, Chapter 7 Evaluation of Eigenvalues, pages 245–247. Springer, 1st edition, Berlin, Springer-Verlag, 2004.
10. C. L. Giles. The future of citeseer. In *10th European Conference on PKDD (PKDD'06)*, page 2, Berlin, Germany, September 2006.
11. Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *Proceedings of the Thirtieth international conference on Very large data bases (VLDB'04)*, pages 576–587. VLDB Endowment, Toronto, Canada, August-September 2004.
12. J. E. Hirsch. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences*, 102:16569, 2005.
13. G. Jeh and J. Widom. SimRank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD conference (KDD'02)*, pages 538–543, ACM, Edmonton, Alberta, Canada, July 2002.
14. W. Jiang, J. Vaidya, Z. Balaporia, C. Clifton, and B. Banich. Knowledge discovery from transportation network data. In *Proceedings of the 21st ICDE Conference (ICDE'05)*, pages 1061–1072, Tokyo, Japan, April 2005.
15. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.
16. B. Long, Z. M. Zhang, X. Wú, and P. S. Yu. Spectral clustering for multi-type relational data. In *ICML'06*, pages 585–592, Pittsburgh, Pennsylvania, USA, June 2006.
17. U. Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.
18. M. E. J. Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20):208701+, October 2002.
19. Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: Bringing order to web objects. In *Proceedings of the fourteenth International World Wide Web Conference (WWW'05)*, pages 567–574. ACM, Chiba, Japan, May 2005.
20. S. Roy, T. Lane, and M. Werner-Washburne. Integrative construction and analysis of condition-specific biological networks. In *Proceedings of AAAI'07*, pages 1898–1899, Vancouver, British Columbia, Canada, July 2007.
21. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

22. A. Sidiropoulos, D. Katsaros, and Y. Manolopoulos. Generalized h-index for disclosing latent facts in citation networks. *CoRR*, abs/cs/0607066, 2006. http://arxiv.org/abs/cs/0607066.
23. Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu. Rankclus: Integrating clustering with ranking for heterogenous information network analysis. In *EDBT'09*, pages 565–576, Saint Petersburg, Russia, March 2009.
24. Y. Sun, Y. Yu, and J. Han. "Ranking-based clustering of heterogeneous information networks with star network schema". In *KDD'09*, pages 797–806, Paris, France, June-July 2009.
25. U. von Luxburg. A tutorial on spectral clustering. Technical report, Max Planck Institute for Biological Cybernetics, 2006.
26. N. Wang, S. Parthasarathy, K.-L. Tan, and A. K. H. Tung. Csv: visualizing and mining cohesive subgraphs. In *SIGMOD'08*, pages 445–458, Vancouver, BC, Canada, June 2008.
27. X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. Scan: a structural clustering algorithm for networks. In *KDD'07*, pages 824–833, San Jose, California, USA, August 2007.
28. O. Zamir and O. Etzioni. Grouper: A dynamic clustering interface to web search results. *Computer Networks*, 31: 1361–1374, 1999.
29. C. Zhai and J. D. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transaction on Information Systems*, 22(2):179–214, 2004.
30. C. Zhai, A. Velivelli, and B. Yu. A cross-collection mixture model for comparative text mining. In *KDD'04*, pages 743–748, Seattle, Washington, USA, August 2004.
31. D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Chapter 18
Mining Large Information Networks by Graph Summarization

Chen Chen, Cindy Xide Lin, Matt Fredrikson, Mihai Christodorescu, Xifeng Yan, and Jiawei Han

Abstract Graphs are prevalent in many domains such as bioinformatics, social networks, Web, and cybersecurity. Graph pattern mining has become an important tool in the management and analysis of complexly structured data, where example applications include indexing, clustering, and classification. Existing graph mining algorithms have achieved great success by exploiting various properties in the *pattern space*. Unfortunately, due to the fundamental role subgraph isomorphism plays in these methods, they may all enter into a pitfall when the cost to enumerate a huge set of isomorphic embeddings blows up, especially in large graphs. The solution we propose for this problem resorts to reduction on the *data space*. For each graph, we build a *summary* of it and mine this shrunk graph instead. Compared to other data reduction techniques that either reduce the number of transactions or compress between transactions, this new framework, called SUMMARIZE-MINE, suggests a third path by *compressing within transactions*. SUMMARIZE-MINE is effective in cutting down the size of graphs, thus decreasing the embedding enumeration cost. However, compression might lose patterns at the same time. We address this issue by generating *randomized* summaries and repeating the process for multiple rounds, where the main idea is that true patterns are unlikely to miss from all rounds. We provide strict probabilistic guarantees on pattern loss likelihood. Experiments on real malware trace data show that SUMMARIZE-MINE is very efficient, which can find interesting malware fingerprints that were not revealed previously.

18.1 Introduction

Recent years have witnessed the prevalence of graph data in many scientific and commercial applications, such as bioinformatics, social networks, Web, and cybersecurity, partly because graphs are able to model the most complex data structures.

C. Chen (✉)
University of Illinois at Urbana-Champaign, Urbana, IL, USA
e-mail: cchen37@uiuc.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_18,

As illustrated by the enhancement made to many core tasks of these domains, e.g., indexing [29] and classification [6, 14], mining graph patterns that frequently occur (for at least min_sup times) can help people get insight into the structures of data, which is well beyond traditional exercises of frequent patterns, such as association rules [1].

However, the emergence of bulky graph data sets places new challenges for graph data mining. For these scenarios, the target graphs are often too large which may severely restrict the applicability of current pattern mining technologies. For example, one emerging application of frequent graph patterns is to analyze the behavior graphs of malicious programs. One can instrument malicious binaries to generate system call graphs, where each node is a system call event. By comparing the subtle differences between graphs generated by malware and benign programs, it is possible to find those graph fingerprints that are common and unique in malicious programs [5]. Unfortunately, due to the bulkiness and complexity of system call graphs, we found that none of the state-of-art mining algorithms can serve this new and critical task well. Similar problems are also encountered for biological networks and social networks.

Existing frequent subgraph mining algorithms, like those developed in [13, 15, 28], achieved great success using strategies that efficiently traverse the *pattern space*; during this process, frequent patterns are discovered after checking a series of subgraph isomorphisms against the database. However, as we argue in this paper, these methods ignore the important fact that isomorphism tests are sometimes expensive to perform. The key issue here is a huge set of isomorphic embeddings that may exist. In order to check the occurrences of a pattern in a large graph, one often needs to enumerate exponentially many subgraphs. This situation is further worsened by the possible overlaps among subgraphs. Looking at $G_1, G_2, \ldots$ in Fig. 18.1, subgraphs such as triangles might share a substantial portion in common, while only one different node/edge would require them to be examined twice, which quickly blows up the total cardinality.

We use a simple example to demonstrate the above scenario. Suppose we have 1,000,000 length-2 paths in a large graph and we would like to check if it has a triangle inside. These 1 million paths have to be checked one by one because each of them has the potential to grow into a full embedding of the triangle pattern. The same dilemma exists for any pattern that includes a length-2 path. Such a huge number of possible embeddings become a severe bottleneck for graph pattern mining tasks.

Now let us consider possible ways to reduce the number of embeddings. In particular, since many embeddings overlap substantially, we explore the possibility of somehow "merging" these embeddings so that the overall cardinality is significantly reduced. As Fig. 18.1 depicts, merging of embeddings is achieved by binding vertices with identical labels into a single node and collapsing the network correspondingly into a smaller version. As suggested by previous studies [3, 25], the above process indeed provides a *graph summary* that generalizes our view on the data to a higher level, which can facilitate analysis and understanding, similar to what OLAP (On-Line Analytical Processing) does for relational databases.

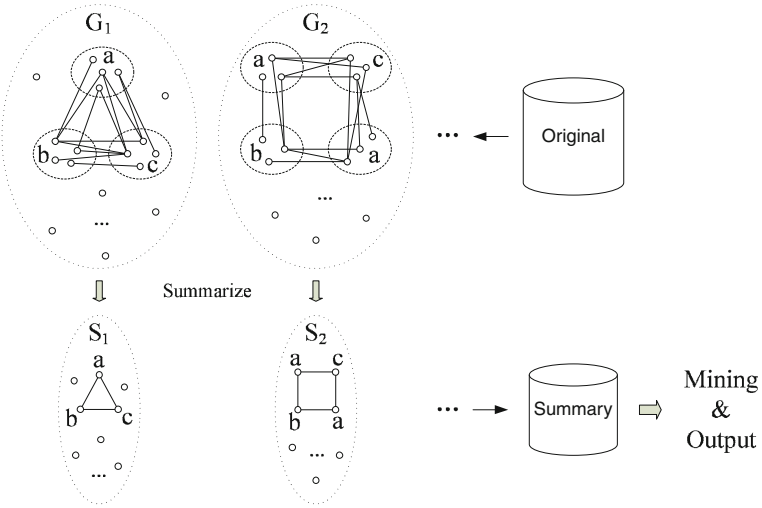


Fig. 18.1 The SUMMARIZE-MINE framework

Graph summarization leads to a dramatic cut-down of graph size as well as the total number of embeddings, which makes subgraph isomorphism cheaper to perform. This forms the main idea of our SUMMARIZE-MINE framework: In Fig. 18.1, we first summarize the original graphs $\{G_1, G_2, \ldots\}$ into small summaries $\{S_1, S_2, \ldots\}$, which are then mined for frequent patterns, where state-of-art algorithms should now perform well. However, the price paid here is the possible loss of patterns, i.e., there could exist false positives and false negatives. For false positives, one can always verify their frequency against the original database and discard those failing ones (interestingly, as we shall discuss later, based on the relationship between G_i and S_i, a lot of verification efforts can be transferred to the small-sized S_i, as well); for false negatives, we choose to generate summaries in a *randomized* manner and repeat the process for multiple rounds. Intuitively, true patterns are unlikely to miss from all rounds.

Recapitulating the above discussions, we outline the contributions made in this chapter as follows.

First, a previously neglected issue in frequent graph pattern mining, i.e., the intrinsic difficulty to perform embedding enumeration in large graphs, is examined, which could easily block many downstream applications. Compared to previous studies that focus on the efficient traversal of *pattern space*, the perspective of this work is *data space* oriented, which leads to an innovative SUMMARIZE-MINE framework. The power and efficiency of our algorithm is validated by extensive experiments on real program analysis data, which can find interesting malware fingerprints that were not revealed previously.

Second, the data reduction principle we adopt is to *compress information within transactions*. It eliminates the shortcoming of lossy summarization by a *randomizing* technique, which repeats the whole process for multiple rounds and achieves

strict probabilistic guarantees. This is novel compared to other methods that either reduce the number of transactions (e.g., sampling [26]) or compress between transactions (e.g., FP-Growth [8] losslessly compresses the whole data set into an FP-tree for frequent itemset mining).

Third, our proposed method of reducing data within transactions supplemented by randomized mechanisms marks an additional dimension that is orthogonal to the state-of-art pattern mining technologies. In this sense, one can freely combine SUMMARIZE-MINE with other optimizations suggested in the past to further enhance their performance, and the idea is not restricted to graphs, which can also be extended to sequences, trees, etc.

Finally, nowadays, extremely huge networks such as those of Internet cyberattacks and on-line social network websites (e.g., Facebook and MySpace) are not uncommon; sometimes, they even cannot fit in main memory, which makes it very hard for people to access and analyze. To this extent, the usage of SUMMARIZE-MINE can be viewed from another perspective: Considering the increasingly important role of summarization as a necessary preprocessing step, we have made a successful initial attempt to analyze how this procedure would impact the underlying patterns (frequent substructures being a special instance). It is crucial for the applications to understand when and to what degree a compressed view can represent the original data in terms of its patterns.

The rest of this chapter is organized as follows. Preliminaries and the overall SUMMARIZE-MINE framework are outlined in Sections 18.2 and 18.3. The major technical investigations, including probabilistic analysis of false negatives, verification of false positives, and iterating multiple times to ensure result completeness, are given in Sections 18.4, 18.5 and 18.6, respectively. Section 18.7 presents experimental results, Section 18.8 discusses related work, and Section 18.9 concludes this study.

18.2 Preliminaries

In this chapter, we will use the following notations. For a graph g, $V(g)$ is its vertex set, $E(g) \subseteq V(g) \times V(g)$ is its edge set, and l is a label function mapping a vertex or an edge to a label.

Definition 1 (Subgraph Isomorphism) For two labeled graphs g and g', a *subgraph isomorphism* is an *injective* function $f : V(g) \rightarrow V(g')$, such that (1) $\forall v \in V(g), l(v) = l'(f(v))$, and (2) $\forall (u, v) \in E(g), (f(u), f(v)) \in E(g')$ and $l(u, v) = l'(f(u), f(v))$, where l and l' are the labeling functions of g and g', respectively. Under these conditions, f is called an *embedding* of g in g', and g is called a *subgraph* of g', denoted as $g \subseteq g'$.

Definition 2 (Frequent Subgraph) Given a graph database $D = \{G_1, G_2, \ldots, G_n\}$ and a graph pattern p, let D_p be the set of graphs in D where p appears as a subgraph. We define the *support* of p as $sup(p) = |D_p|$, whereas D_p is referred

as p's *supporting graphs* or p's *projected database*. With a predefined threshold min_sup, p is said to be *frequent* if $sup(p) \geq min\_sup$.

The problem targeted in this paper is essentially the same as that of a well-studied graph mining task: finding all frequent subgraphs in a database D, except that the graphs in D are now associated with large size. As we mentioned in the introduction, our proposal is to perform summarization at first.

Definition 3 (Summarized Graph) Given a labeled graph G, suppose its vertices $V(G)$ are partitioned into groups, i.e., $V(G) = V_1(G) \cup V_2(G) \cup \cdots \cup V_k(G)$, such that (1) $V_i(G) \cap V_j(G) = \phi$ $(1 \leq i \neq j \leq k)$, and (2) all vertices in $V_i(G)$ $(1 \leq i \leq k)$ have the same labels. Now, we can *summarize* G into a compressed version S, written as $S \prec G$, where (1) S has exactly k nodes $v_1, v_2, \ldots, v_k$ that correspond to each of the groups (i.e., $V_i(G) \mapsto v_i$), while the label of v_i is set to be the same as those vertices in $V_i(G)$, and (2) an edge (v_i, v_j) with label l exists in S if and only if there is an edge $(u, \hat{u})$ with label l between some vertex $u \in V_i(G)$ and some other vertex $\hat{u} \in V_j(G)$.

Based on the above definition, *multi-edge* becomes possible for a summarized graph, i.e., there might be more than one labeled edge that exist between two vertices $v_i, v_j \in V(S)$, if there is an edge $(u_1, \hat{u_1})$ with label l_1 and another edge $(u_2, \hat{u_2})$ with label $l_2 \neq l_1$ such that u_1, u_2 is in the node group $V_i(G)$ and $\hat{u_1}, \hat{u_2}$ is in the node group $V_j(G)$. To find patterns on top of such summaries, slight modifications are needed because traditional graph mining algorithms in general assume *simple graphs* (i.e., no self-loops and multi-edges). We shall get back to this issue later as the discussion proceeds.

18.3 The SUMMARIZE-MINE Framework

Given a graph database $D = \{G_1, G_2, \ldots, G_n\}$, if we summarize each $G_i \in D$ to $S_i \prec G_i$, then a summarized database $D' = \{S_1, S_2, \ldots, S_n\}$ is generated. Denote the collection of frequent subgraphs corresponding to D and D' as $FP(D)$ and $FP(D')$, respectively. In this section, we are going to examine the relationship between these two pattern sets and investigate the possibility to shift mining from D to D'.

Intuitively, we expect that $FP(D)$ and $FP(D')$ are similar to each other if the summarization from D to D' is properly conducted. As for the portion that is different between them, there are two cases.

Definition 4 (False Negatives) A subgraph p frequent in D but not frequent in D', i.e., $p \in \text{FP}(D)$ and $p \notin \text{FP}(D')$, is called a *false negative* caused by summarization.

Definition 5 (False Positives) A subgraph not frequent in D but frequent in D', i.e., $p \notin \text{FP}(D)$ and $p \in \text{FP}(D')$, is called a *false positive* caused by summarization.

For the rest of this section, we are going to discuss how these two types of errors can be remedied, which finally gives rise to a novel SUMMARIZE-MINE framework.

18.3.1 Recovering False Negatives

False negatives include those patterns that are missed out after we summarize the graphs. In Fig. 18.2, we explain the reason behind. Suppose p is a graph pattern such that $p \subseteq G_i$, and correspondingly f is an embedding of p in G_i. Consider the summary $S_i \prec G_i$; f will disappear if there exist two nodes $u, \hat{u} \in V(p)$ whose images in $V(G_i)$, i.e., $f(u)$ and $f(\hat{u})$, are merged together as we shrink G_i into S_i. This will cause the support of p to decrease upon summarization.

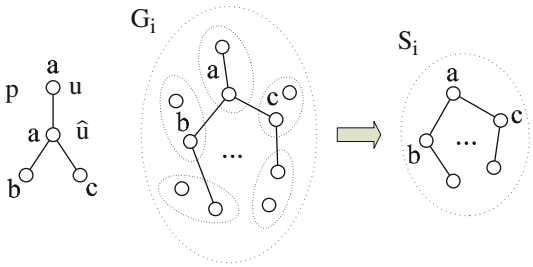


Fig. 18.2 The cause of false negatives

So, how should we avoid false negatives? To begin with, it is easy to prove the following lemma.

Lemma 1 *For a pattern p, if each of its vertices bears a different label, then p's supporting graph set in the summarized database D' is no smaller than that in the original database D, i.e., $D'_p \supseteq D_p$.* ■

Proof Suppose $G_i \in D_p$, i.e., $p \subseteq G_i$; let f be an embedding of p in G_i. Obviously, for p's vertices $u_1, \ldots, u_m$, their corresponding images $f(u_1), \ldots, f(u_m)$ in G_i must have different labels, and thus $f(u_1), \ldots, f(u_m)$ should belong to m separate groups, which end up as distinct nodes $v_1, \ldots, v_m$ in the summarized graph $S_i \prec G_i$. Define another injective function $f' : V(p) \rightarrow V(S_i)$ by mapping u_j to v_j $(1 \leq j \leq m)$. Based on Definition 3, it is easy to verify that whenever there is an edge $(u_{j_1}, u_{j_2}) \in E(p)$ with label l, there exists a corresponding edge $(v_{j_1}, v_{j_2}) \in E(S_i)$ bearing the same label. Now, f' represents a qualified embedding of p in S_i. More generally, $p \subseteq S_i$ will hold for each G_i's shrunk version S_i if $G_i \in D_p$, indicating that D'_p is at least as large as D_p.

Based on Lemma 1, false negatives can only happen for those patterns with at least two identically labeled vertices. Meanwhile, from the proof above, we conclude that even if two vertices $u_1, u_2 \in V(p)$ possess the same label, as long as their images $f(u_1)$, $f(u_2)$ are not merged by summarization, the embedding f is still preserved. According to these observations, we could partition nodes into identically labeled groups on a *random* basis, where for those vertices with same labels in pattern p, they have a substantial probability $q(p)$ to stay in different groups, which guarantees that no embeddings will be destroyed. Facing such probabilistic pattern loss, we decide to deliberately lower the support threshold in the summarized

database by a small margin to $min\_sup' < min\_sup$: As we shall prove in Section 18.4, this will then insure a high probability P for patterns to remain frequent in D'. Finally, to further reduce the false-negative rate, we can perform randomized summarization for multiple times in an independent fashion, because the overall pattern missing probability $(1 - P)^t$ will quickly converge to 0 as the number of iterations t increases. The details of false-negative analysis are given in Section 18.4.

18.3.2 Discarding False Positives

Given a graph pattern p, there is also possibility for its support to increase upon summarization. Figure 18.3 shows a "faked" embedding of p formed in the summarized graph S_i, where two sets of edges originally adjacent to different vertices with label a are now attached to the same node.

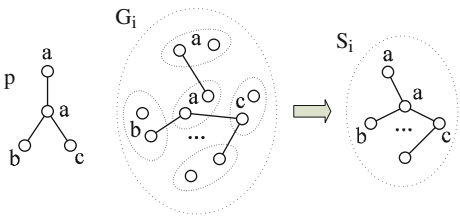


Fig. 18.3 The cause of false positives

It is much easier to deal with false positives. For false negatives, we must provide a mechanism to recover true patterns that have disappeared after summarization, while for false positives, we only need an efficient *verification* scheme to check the result set and get rid of those entries that are actually infrequent.

A straightforward verification scheme computes the support of every $p \in FP(D')$ in the original database D: If $sup(p)$ is smaller than min_sup, we discard p from the output. Interestingly, there is a better way to verify patterns by leveraging the summaries: The embedding of p in the summarized graph actually reveals its possible locations in the original graph, which can be used to speed up the process. Technical details will be covered in Section 18.5.

18.3.3 The Overall Algorithm Layout

With randomization and verification, the SUMMARIZE-MINE framework is outlined as follows.

1. *Summarization*: For each G_i in a graph database D, randomly partition its vertex set $V(G_i)$: For vertices with label $l_j (1 \leq j \leq L)$, where L represents the total number of labels, we assign x_j groups. This will result in x_j nodes with label l_j

in the corresponding summary graph. As the application characteristics vary, we can control the summarization process by changing $x_1, \ldots, x_L$ to best cope with the situation.
2. *Mining*: Apply any state-of-art frequent subgraph mining algorithm on the summarized database $D' = \{S_1, S_2, \ldots, S_n\}$ with a slightly lowered support threshold min_sup', which generates the pattern set $FP(D')$.
3. *Verification*: Check patterns in $FP(D')$ against the original database D, remove those $p \in FP(D')$ whose support in D is less than min_sup, and transform the result collection into R'.
4. *Iteration*: Repeat steps 1–3 t times. To guarantee that the overall probability of missing any frequent pattern is bounded up by ε, we set the number of rounds t as $\lceil \frac{\log \varepsilon}{\log(1-P)} \rceil$, where $1 - P$ is the false-negative rate in one round.
5. *Result combination*: Let $R'_1, R'_2, \ldots, R'_t$ be the patterns obtained from different iterations; the final result is $R' = R'_1 \cup R'_2 \cup \cdots \cup R'_t$.

Compared to the true pattern set R that would be mined from the original database D if there are enough computing resources, no false positives exist, i.e., $R' \subseteq R$, and the probability for a pattern $p \in R$ to miss from R' is at most ε. Note that the verification step here is put after the mining step for clarity purposes. As we shall see later, these two steps can also be interleaved, where verifications are performed on the fly: Whenever a pattern p is discovered, SUMMARIZE-MINE verifies it immediately if p has not been discovered and verified by previous iterations.

In the following, we will start from probabilistic analysis of false negatives in Section 18.4, followed by Section 18.5, which focuses on the verification of false positives, and Section 18.6, which discusses iterative SUMMARIZE-MINE as well as result combination.

18.4 Bounding the False-Negative Rate

As we proved in Lemma 1 of Section 18.3.1, for a pattern p and a graph G_i in the original database, if p is a subgraph of G_i through embedding f, then as G_i is summarized into S_i, f disappears from S_i if $f(u_1), \ldots, f(u_m)$ are disseminated into less than m groups and thus correspond to less than m vertices in S_i. Suppose there are m_j and x_j vertices with label l_j in p and S_i, respectively, we have the following lemma.

Lemma 2 *For a graph pattern p, if $p \subseteq G_i$, then p is also a subgraph of $S_i \prec G_i$ with probability at least*

$$\frac{P_{x_1}^{m_1} \cdots P_{x_L}^{m_L}}{x_1^{m_1} \cdots x_L^{m_L}},$$

given that the grouping and merging of nodes that transform G_i into S_i is performed on a completely random basis. Here, $P_{x_j}^{m_j}$ represents the number of permutations, which is equal to $\binom{x_j}{m_j} m_j!$

Proof Consider an embedding f through which p is a subgraph of G_i. The probability that all m_j vertices with label l_j are assigned to m_j different groups (and thus f continues to exist) is

$$\frac{x_j}{x_j}\frac{x_j-1}{x_j}\cdots\frac{x_j-m_j+1}{x_j}=\frac{P_{x_j}^{m_j}}{x_j^{m_j}}.$$

Multiplying the probabilities for all L labels (because the events are independent), we have

$$\text{Prob}\big[p\subseteq S_i\big]\geq\frac{P_{x_1}^{m_1}\cdots P_{x_L}^{m_L}}{x_1^{m_1}\cdots x_L^{m_L}}.$$

Here, x_j must be at least as large as m_j to make the product of probabilities meaningful, and during implementation, there is often no problem for us to make $x_j >> m_j$ so that vertices with identical labels will not collide with high probability. ■

To simplify analysis, if we stick with a particular set of x_j's ($1\leq j\leq L$) when summarizing different graphs in the database, the probability bound in Lemma 2 can be written as $q(p)$, since its value only depends on the label distribution of pattern p, which holds for any $G_i\in D_p$. Now, because of those embeddings that disappear due to summarization, it is well expected that the pattern support will experience some drop, with Theorem 1 characterizing the probability of seeing a particular dropping magnitude.

Theorem 1 *Suppose a pattern p's support in the original database is s, i.e., $|D_p|=s$, for any $s'\leq s$, the probability that p's support in D' falls below s' upon summarization can be bounded as follows:*

$$\text{Prob}\big[|D'_p|\leq s'\big]\leq\sum_{T=0}^{s'}\binom{s}{T}q(p)^T[1-q(p)]^{s-T}.$$

Proof For each $G_i\in D_p$, we focus on a particular subgraph embedding f_i and define an indicator variable I_i such that $I_i=1$ if f_i continues to exist in S_i and $I_i=0$ otherwise. Then,

$$\text{Prob}\big[|D'_p|>s'\big]\geq\text{Prob}\Big[\sum_{G_i\in D_p}I_i>s'\Big],$$

because whenever $\sum_{G_i \in D_p} I_i > s'$, there must be more than s' subgraph embeddings that are preserved in the summarized database and thus $|D'_p| > s'$. We have

$$\text{Prob}\big[|D'_p| \le s'\big] \le \text{Prob}\Big[\sum_{G_i \in D_p} I_i \le s'\Big]$$
$$= \sum_{T=0}^{s'} \text{Prob}\Big[\sum_{G_i \in D_p} I_i = T\Big].$$

The difference between the left- and right-hand side probabilities is due to three effects: (1) there could be multiple embeddings of p in G_i, so that p's support after summarization may not decrease even if one embedding disappears, (2) a "faked" embedding like that depicted in Fig. 18.3 might emerge to keep p as a subgraph of S_i, and (3) "faked" embeddings can also happen for a graph G_j which originally does not contain p. Now, because events are independent,

$$\text{Prob}\left[\sum_{G_i \in D_p} I_i = T\right] = \binom{s}{T} q(p)^T [1 - q(p)]^{s-T},$$

where $q(p) = \text{Prob}\big[I_i = 1\big]$ for all i such that $G_i \in D_p$. Finally,

$$\text{Prob}\big[|D'_p| \le s'\big] \le \sum_{T=0}^{s'} \text{Prob}\Big[\sum_{G_i \in D_p} I_i = T\Big]$$
$$= \sum_{T=0}^{s'} \binom{s}{T} q(p)^T [1 - q(p)]^{s-T}$$

is proved. ■

Corollary 1 *Assume that the support threshold is min_sup, we set a new threshold $min\_sup' < min\_sup$ for the database D' summarized from D and mine frequent subgraphs on D'. The probability for a pattern p to be a false negative, i.e., p is frequent in D but not frequent in D', is at most*

$$\sum_{T=0}^{min\_sup'-1} \binom{min\_sup}{T} q(p)^T [1 - q(p)]^{min\_sup-T}.$$

Proof Being a false negative, we have $s = |D_p| \ge min\_sup$ and $|D'_p| \le min\_sup' - 1$. Let $s' = min\_sup' - 1$; a direct application of Theorem 1 leads to

$$\text{Prob}\big[|D'_p| < min\_sup'\big] \leq \sum_{T=0}^{min\_sup'-1} \binom{s}{T} q(p)^T [1-q(p)]^{s-T},$$

where the right-hand side corresponds to a binomial random variable $B\big(s, q(p)\big)$'s cumulative distribution function (CDF) being evaluated at s'. Denote the CDF of a binomial variable $Y \sim B(N, p)$ as $F_B(N, p; n) = \text{Prob}\big[Y \leq n\big]$, we have $F_B(N, p; n)$ monotonically decreasing in N, because Y is the sum of N independent Bernoulli random variables $X_1, \ldots, X_N \sim \text{Ber}(p)$: When more X_i's get involved, it is naturally harder to have their sum Y bounded up by some fixed number n. This leads to $F_B(N_1, p; n) \geq F_B(N_2, p; n)$ if $N_1 \leq N_2$. Finally, since $s \geq min\_sup$,

$$\begin{aligned} & F_B\big(s, q(p); s'\big) \leq F_B\big(min\_sup, q(p); s'\big) \\ = & \sum_{T=0}^{min\_sup'-1} \binom{min\_sup}{T} q(p)^T [1-q(p)]^{min\_sup-T}, \end{aligned}$$

which is combined with the inequality at the beginning to complete the proof.

As the SUMMARIZE-MINE framework suggests, the false-negative rate after t iterations is $(1-P)^t$. To make $(1-P)^t$ less than some small ε, one can either increase the number of rounds t or decrease the one-round false-negative rate $1-P$, which is achieved by lowering the support threshold min_sup' on the summarized database D'. Since increasing t and reducing min_sup' will both lead to a longer mining time, we could simultaneously control both parameters to find an optimal trade-off point where the best efficiency is achieved. This will be tested in the experiments.

18.5 Verifying False Positives

To implement SUMMARIZE-MINE, we take gSpan [28] as the skeleton of our mining algorithm. The main idea of gSpan is as follows: Each labeled graph pattern can be transformed into a sequential representation called *DFS code*, based on a depth-first traversal of the pattern. With a defined *lexicographical order* on the DFS code space, all subgraph patterns can be organized into a tree structure, where (1) patterns with k edges are put on the kth level and (2) a preorder traversal of this tree would generate the DFS codes of all possible patterns in the lexicographical order. Figure 18.4 shows a pattern tree, where $v_1, v_2, \ldots, v_n$ are vertex patterns, p_1 is a pattern with one edge, and p_1 is a subgraph of p_2.

This DFS code-based pattern tree is used in SUMMARIZE-MINE. For each graph pattern p, we conduct the following steps.

1. We decide whether the DFS code of p is minimum according to the defined lexicographical order. Here, patterns might have different codes in the tree because of graph isomorphisms but we only need to examine one of them. In this sense,

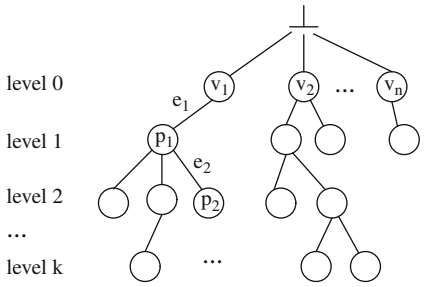


Fig. 18.4 A pattern tree

non-minimum DFS codes can be discarded since the corresponding minimum ones must have been visited by the preorder traversal.
2. We check p against the summarized graphs and get p's projected database D'_p. If $|D'_p|$ falls below min_sup', we abort the search along this branch.
3. For each summary $S_i \in D'_p$, we enumerate all embeddings of p in S_i and based on that determine every possible one-edge extension that can be added to them. These candidate patterns are inserted into the pattern tree, which will be explored later.
4. When we drive search into a particular branch, D'_p is passed down as a transaction ID-list, which will help pruning since the new projected database there can only be a subset of D'_p.

If SUMMARIZE-MINE generates a pattern tree as shown in Fig. 18.4, we could start checking false positives from big patterns so that the verification of many smaller patterns can be avoided. Given two patterns p_1 and p_2, where p_1 is a subgraph of p_2, there is no need to verify p_1 if p_2 already passes the min_sup threshold, because $sup(p_1) \geq sup(p_2) \geq min\_sup$. Referring to the DFS code tree, this is done by visiting the data structure in a *bottom-up* manner, which can be easily implemented through a postorder traversal. On the other hand, it seems that adopting the opposite direction, i.e., visiting the tree in a *top-down* manner through a preorder traversal, might also give us some advantages: If we verify p_1 before p_2, then there is no need to further try p_2 if p_1 already fails the test, since $min\_sup > sup(p_1) \geq sup(p_2)$. In this sense, considering the question of picking a better one from these two approaches, it really depends on how many false positives exist in the set of patterns we want to verify, which could be data-specific. Generally speaking, if there are not/too many false positives, the bottom-up/top-down approach should work well.

Summary-Guided Isomorphism Checking. During the verification process, after getting D'_p, we want to check a pattern p against each $G_i \in D$ and get its support in the original database. Suppose $G_i \succ S_i$, there are two cases: $S_i \in D'_p$ and $S_i \notin D'_p$. For the first case, the embedding of p in S_i could help us quickly find its possible embeddings in G_i. Let $f : V(p) \rightarrow V(S_i)$ be the embedding of p

in S_i, where the images of p's vertices under f are $f(u_1), \ldots, f(u_m)$. Recall that G_i is summarized into S_i by merging a node group of G_i into a single vertex of S; we can check whether there exists a corresponding embedding of p in G_i by picking one node from each of the node groups that have been summarized into $f(u_1), \ldots, f(u_m)$ and examining their mutual connections. This should be more efficient than blindly looking for a subgraph isomorphism of p in G_i, without any clue about the possible locations. For the second case, there is no embedding of p in S_i to leverage, can we also confirm that $p \nsubseteq G_i$ by looking at the summary only? Let us choose a subgraph $p' \subseteq p$ such that each of p''s vertices bears a different label, and test p' against S_i; based on Lemma 1, since the embeddings of p' can never be missed upon summarization, if we can confirm that $p' \nsubseteq S_i$, then it must be true that $p' \nsubseteq G_i$ and there is no hope for p, a supergraph of p', to exist in G_i, either. Concerning implementation, we can always make p' as big as possible to increase the pruning power. Finally, we have transformed isomorphism tests against the original large graph G_i to its small summary S_i, thus taking advantage of data reduction.

18.6 Iterative SUMMARIZE-MINE

In this section, we combine the summarization, mining, and verification procedures together and put them into an iterative framework. As discussed previously, adding more iterations can surely reduce the probability of false negatives; however, it introduces some problems, as well. For example, the final step of SUMMARIZE-MINE is to merge all R'_k's ($k = 1, 2, \ldots, t$) into a combined set R', which requires us to identify what the individual mining results have in common so that only one copy is retained. Furthermore, due to the overlap among $R'_1, R'_2, \cdots$, a large number of patterns might be repeatedly discovered and verified.

One solution to this problem is to represent the patterns in each R'_k by their DFS codes, which are then sorted in lexicographical order, facilitating access and comparison. However, this approach is still costly. Our proposed strategy is as follows: Since $R'_1, R'_2, \ldots$ are mined from successive random summarizations of the original graph database D, it is expected that R'_k's would not be too different from each other because they are all closely related to $FP(D)$, the "correct" set of frequent subgraphs that would be mined from D. This hints us to unify the mining process of different iterations into a single data structure, i.e., use only one pattern tree $\mathcal{T}$ to drive the mining ahead.

The advantages of doing so are twofold. First, if a single data structure is maintained, and we incrementally modify $\mathcal{T}$ (i.e., recover false negatives that have been wrongly omitted by previous rounds) as the mining of multiple iterations proceeds, then the problem of merging $R'_1, R'_2, \ldots$ is automatically solved, because there is only one pattern tree, which stores the combined result. Second, in such an integrated manner, intermediate calculations achieved in earlier rounds may help the processing of later rounds, which cannot be achieved if consecutive iterations are separated. The below example demonstrates this.

Our setting is as follows: In round 1, a pattern tree holding R'_1 is generated, which is drawn on the left-hand side of Fig. 18.5, i.e., $\mathcal{T}_1$. Then, as we go into round 2, some patterns missed from the first iteration will be recovered (specifically, p_3 and p_4), which update $\mathcal{T}_1$ into a new tree $\mathcal{T}_2$ that is drawn on the right-hand side.

Now, suppose we have finished round 1 and verified the patterns in R'_1, e.g., p_1, p_2, by checking their support against D, the corresponding projected databases, i.e., D_{p_1}, D_{p_2}, become known to us. These two support sets, represented as ID-lists, are stored with tree nodes p_1, p_2 for later use. Note that, since ID-lists only record integer identifiers of the transaction entries, they have moderate size and can be easily maintained/manipulated. The same strategy has been widely used in other graph data management tasks, e.g., indexing [29].

Moving on to round 2, we start from the tree's root p_1 (note that, although p_1 has been verified and shown to be frequent by round 1, we cannot bypass it in round 2, because patterns such as p_3 and p_4, which are infrequent in round 1 but frequent in round 2, have to be grown from it), where the first thing to do is checking p_1's support against the second-round summarized database $D'^2 = \{S_1^2, S_2^2, \ldots, S_n^2\}$. Interestingly, it is only necessary to test p_1 with regard to those graphs in D_{p_1}, i.e., what we finally obtain could be a subset of $D'^2_{p_1}$ that is confined within D_{p_1}, i.e., $D'^2_{p_1} \cap D_{p_1}$. This turns out to be OK: For any pattern p_* that would be subsequently grown along the branch of p_1, $p_1 \subseteq p_* \Rightarrow D_{p_*} \subseteq D_{p_1}$ because of the Apriori principle; thus, when p_* is finally verified against the original database D, its support

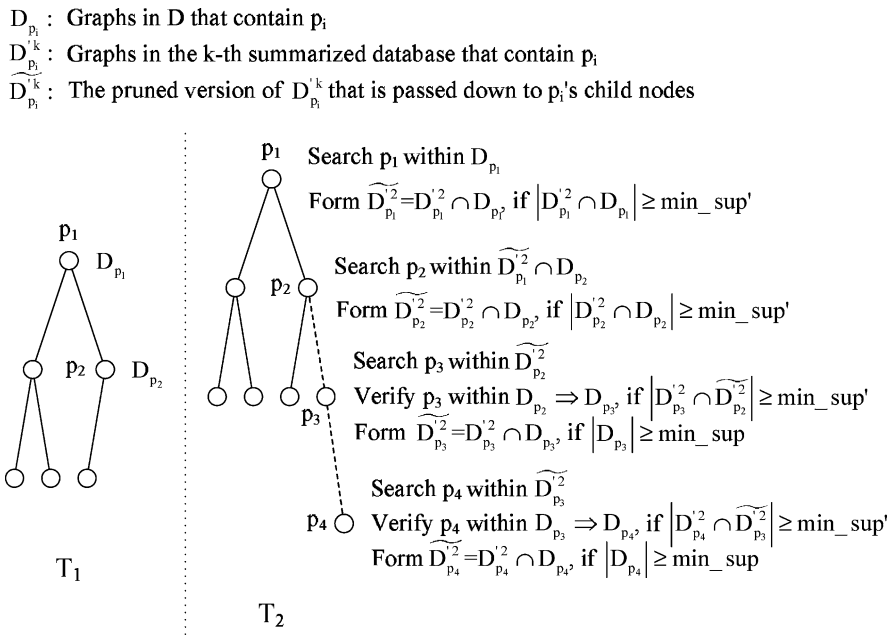


Fig. 18.5 The first two iterations of SUMMARIZE-MINE *with* verified ID-lists

graphs D_{p_*} will be confined within D_{p_1} anyway. This means that an early pruning by the ID-list of D_{p_1}, which is readily available after round 1, should not have any impact on the rest of the mining process.

We draw a few more steps in Fig. 18.5 regarding the utilization of pre-verified ID-lists when it comes to patterns p_3, p_4, and the corollary below proves that the optimizations proposed would not change any theoretical foundation of SUMMARIZE-MINE.

Corollary 2 *The probability bound developed in Corollary 1 still holds if verified ID-lists are used to prune the mining space.*

Proof Given a pattern p, the bound in Corollary 1 is directly related to whether p's embeddings for each of its support graphs in the original database D, i.e., D_p, would diminish or not upon randomized summarizations. As shown above (think p as p_1 in Fig. 18.5), when we utilize verified ID-lists for optimizations, entries in D_p are not filtered out for sure, which means that all deductions made in the corresponding proofs now continue to hold. ■

The described pruning techniques should be applied as early as possible in order to boost performance. In this sense, we shall start verification right after a new pattern is discovered, so that its ID-list might be used even in the current iteration. Putting everything together, the pseudocode of SUMMARIZE-MINE is given in Algorithm 1.

To start Algorithm 1, we call sMine(p_1, D), where p_1 is the root of the pattern tree and D includes every graph in the database. In order to grow all possible patterns, p_1 should be a null graph with zero vertices, as Fig. 18.4 depicts. In line 1, we return immediately if the same p has been shown to be infrequent by previous iterations; and the reason for setting such a flag is to guarantee that unsuccessful verifications will not be performed repeatedly. Line 2 checks whether a given DFS code is minimum. Lines 3 and 4 conduct a pre-pruning if p has been verified in the past and thus an ID-list is readily available. Lines 5–7 proceed like normal frequent subgraph mining algorithms, where support is computed by checking isomorphic embeddings against the current projected database, and all possible one-edge extensions of p are recorded during this process. If the support does not pass the lowered min_sup' threshold on summarized databases, the algorithm returns immediately (line 8); otherwise, we verify p (line 10) if it has not been verified so far (line 9), mark $p.err$ as $true$, and return if p cannot pass the min_sup threshold after verification (line 11). If p indeed can pass min_sup (line 12), we store the ID-list (line 13) and use it to immediately prune the projected database (line 14) that will be passed on when sMine is called for those patterns grown from p (lines 15–17).

We discuss some variations of Algorithm 1 in the following. First, we have been verifying patterns in the same order as they are mined out, which corresponds to a top-down scheme. As we mentioned in Section 18.5, the verified ID-lists of each node in the pattern tree can also be obtained in a bottom-up manner, while the only

Algorithm 1 SUMMARIZE-MINE with verified ID-lists

$D'^k = \{S_1^k, \ldots, S_n^k\}$: The kth-round summarized database.
p: The graph we are visiting on the pattern tree.
PD: Projected database passed from the caller.
$p.err$: A flag stored with p, it equals *true* if p has already failed to pass a verification test in previous iterations.
$p.IDs$: The ID-list stored with p, it equals ϕ if p is discovered for the first time and thus has not been verified.

```
sMine(p, PD) {
1:  if p.err == true then return;
2:  if p's DFS code is not minimum then return;
3:  if p.IDs ≠ φ then PD' = PD ∩ p.IDs;
4:  else PD' = PD;
5:  foreach graph ID i ∈ PD' do
6:      if p ⊈ S_i^k then PD' ← PD' − {i};
7:      else enumerate the embeddings of p in S_i^k; *
8:  if |PD'| < min_sup' then return;
9:  else if p.IDs == φ then
10:     verify p against the original database D;
11:     if |D_p| < min_sup then p.err = true; return;
12:     else
13:         store the IDs according to D_p in p.IDs;
14:         PD' = PD' ∩ p.IDs;
15: foreach p' ∈ pGrow do
16:     if p' is not a child of p in T then insert p' under p;
17:     sMine(p', PD');
}
```

*Steps 4–6 enumerate all possible one-edge extensions that can be made to p, which we denote as $pGrow$.

shortcoming here is that pruning must be delayed until the next iteration, because bottom-up checking can only happen when the whole pattern tree is ready. Second, there are costs, as well as benefits, to calculate and store the exact IDs of every pattern's support graphs in D. Interestingly, suppose we choose bottom-up verification; then for a pattern tree $\mathcal{T}$, we could have only verified those leaf nodes, while all internal nodes are guaranteed to be frequent (because they have even higher support), without any calculations. Thus, it is not always necessary to maintain the ID-lists.

Consider whether or not to compute verified ID-lists, plus that both top-down and bottom-up verification schemes can be selected; there are four cases in total.

- ID-list+top-down: It corresponds to Algorithm 1.
- ID-list+bottom-up: Here, though verification result can only be used in the next iteration, we are not sure whether this shortcoming can be overcome by the relative edge if bottom-up verification is faster than its top-down counterpart. We will reexamine this issue in experiments.

- No ID-list+top-down: This scenario does not make much sense, because in top-down verification, the ID-lists of all patterns in the tree can be obtained as a by-product. So, why not take this "free lunch" to boost performance?
- No ID-list+bottom-up: Fig. 18.6 illustrates the situation. We adopt bottom-up postorder traversal to verify false positives, while successive iterations are essentially independent of each other, except that they share the same tree $\mathcal{T}$ to hold the mining result.

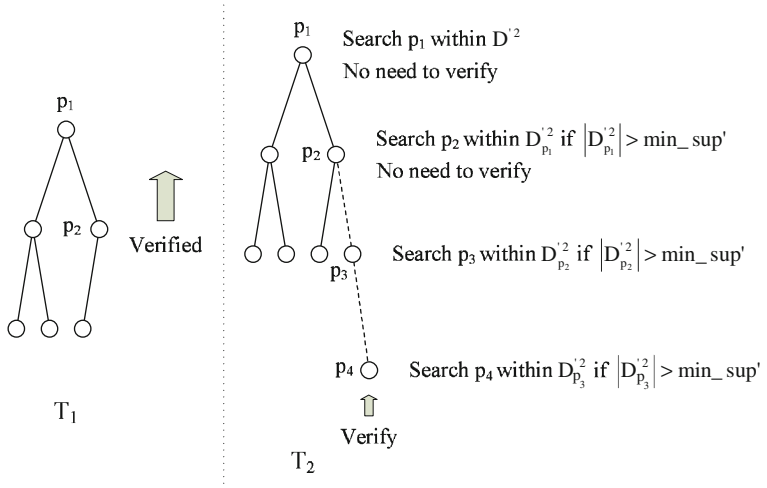


Fig. 18.6 The first two iterations of SUMMARIZE-MINE *without* verified ID-lists

18.7 Experimental Results

In this section, we will provide empirical evaluations of SUMMARIZE-MINE. We have two kinds of data sets: a real data set and a synthetic data set. To be more concrete, we shall use the real data set to show the effectiveness and efficiency of our algorithm, while the synthetic data set will demonstrate the parameter setting mechanism, as well as the method's scalability. All experiments are done on a Microsoft Windows XP machine with an Intel Core 2 Duo 2.5G CPU and 3 GB main memory. Programs are written in Java.

The mining process works by randomly summarizing a graph database, finding patterns from the summaries, and then verifying obtained patterns. As we briefly discussed in Section 18.2, to handle edges with multiple labels during the mining step, we modify gSpan and store a label list with each edge in the graph: A pattern matching will be successful as long as the pattern's corresponding edge label is covered by this list. For the verification step, we shall try alternative schemes

(e.g., top-down, bottom-up), and the optimization that leverages summary-guided isomorphism checking (see Section 18.5) will be adopted by default.

18.7.1 Real Data Set

Program Analysis Data. Program dependence graphs appear in software-security applications that perform characteristic analysis of malicious programs [5]. The goal of such analysis is to identify subgraphs that are common to many malicious programs, since these common subgraphs represent typical attacks against system vulnerabilities, or to identify contrast subgraphs that are present in malicious programs but not in benign ones, since these contrast subgraphs are useful for malware detection. In our experience and as reported by anti-malware researchers, these representative program subgraphs have less than 20 vertices.

We collected dependence graphs from six malware families, including W32.Virut, W32.Stration, W32.Delf, W32.Ldpinch, W32.Poisonivy, and W32.Parite. These families exhibit a wide range of malicious behaviors, including behaviors associated with network worms, file-infecting viruses, spyware, and backdoor applications. In a dependence graph, vertices are labeled with program operations of interest and the edges represent dependency relationships between operations. For example, when the operations are system or library calls, then an edge with label $y = f(x)$ between two vertices v_1 and v_2 captures the information that the system call at v_1 assigns the variable x and the second system call uses the variable y whose value is derived from x. Such dependence graphs are quite large in practice, sometimes with vertex counts up to 20, 000 and edge counts an order of magnitude higher (up to 220, 000 based on our observation). For the experiment data we use, the average number of nodes for all graphs is around 1,300.

Before we move on, let us assume for now that all parameters in Section 18.7.1 are already set to the optimal values. Detailed discussions on how this is achieved will be covered in Section 18.7.2.

Figure 18.7 shows a graph pattern discovered from the Stration family of malware. Stration is a family of mass-mailing worms that is currently making its way across the Internet. It functions as a standard mass-mailing worm by collecting email addresses saved on a hostand sending itself to the recipients, which does display

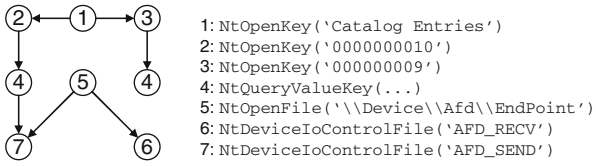


Fig. 18.7 A sample malware pattern

some characteristics of spyware as shown in the figure. The displayed signature corresponds to a malware reading and leaking certain registry settings related to the network devices.

In Fig. 18.8, we plot the probability bound predicted in Theorem 1 against the empirical event frequency that is observed in experiments. Suppose there are X_j nodes with label l_j in a graph $G_i \in D$, we set x_j as $round\left(a_i \cdot \frac{X_j}{\sum_{j=1}^{L} X_j}\right)$, where a_i is the number of nodes to be kept for each database graph. In this way, labels that appear more often in the original graphs will also have more presence in their summarized versions, which is reasonable. Let A be the average number of nodes for graphs in the original database and a be the corresponding number after summarization; the summarization ratio is defined as $\alpha = A/a$. We set $min\_sup = 55\%$ (note that, for a graph data set with big transaction size, min_sup is often set relatively high since small structures are very easy to be contained by a large graph; thus, there would be too many patterns if the support threshold is low), $min\_sup' = 45\%$, $\alpha = 8$, and randomly pick 300 patterns from the output of iteration 1. For each pattern p, we count its support $s = |D_p|$ in the original database D, compute $q(p)$ based on the distribution of p's vertex labels according to Lemma 2, and fix $s' = 70\% \cdot s$ to calculate the theoretical guarantee of $\text{Prob}\left[|D'_p| \leq s'\right]$ as given in the right-hand side of Theorem 1, which is drawn on the x-axis. Then, we further generate 100 copies of D' based on randomized summarization, obtain the percentage of times in which p's support $|D'_p|$ really falls below s', and draw it on the y-axis. Patterns whose vertices are all associated with distinct labels have been omitted, because they can never miss.

It can be seen that our probabilistic guarantee is quite safe, where only very few points exist whose empirical frequencies go beyond the corresponding theoretical bounds, which is possible, because the frequency values calculated by such random sampling may not represent true probabilities. On the other hand, it also shows that real false-negative rate is often not that high. So, we probably do not have to be too conservative when setting the new support threshold min_sup', due to the three effects we pointed out in the proof of Theorem 1.

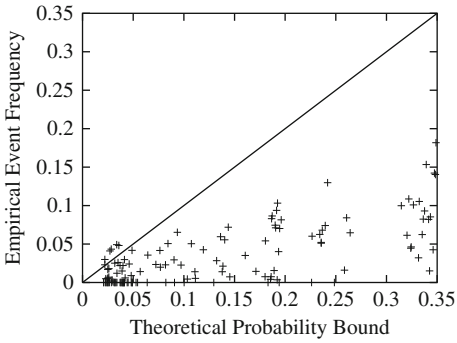


Fig. 18.8 Theoretical guarantee

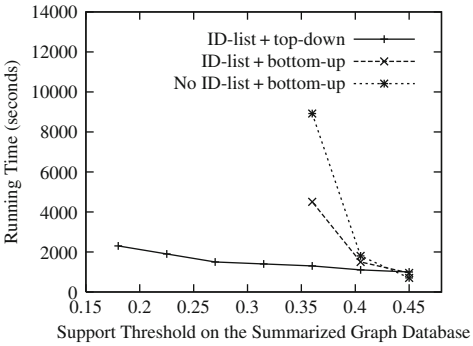


Fig. 18.9 Three alternative strategies

In Fig. 18.9, we draw the running time with regard to min_sup' after fixing $min\_sup = 55\%$, $\alpha = 8$, and compare relative performances of the three strategies we proposed in Section 18.6. Here, two iterations are processed, while one can also increase the number of rounds t to further bring down the pattern miss rate. Based on the testing results, it seems that we are better off using verified ID-lists, because they are very effective in pruning false positives. Suppose a pattern p is mined from D' and after verifying it against D we find that p's support in the original database is less than min_sup, then for ID-list+top-down, we will terminate immediately without growing to p's supergraphs. However, considering No ID-list+bottom-up, as long as the support of these supergraphs in D' is greater than min_sup', they will all be generated and then verified as a batch at the end of each iteration. The advantage of such pre-pruning starts to prevail when min_sup' becomes smaller, which induces more false positives. Based on similar reasoning, the curve for ID-list+bottom-up turns out to appear in the middle, since pruning cannot happen in the first round but it can act in the second round. Finally, due to its general superiority, for the rest of this section, we shall use ID-list+top-down as our default implementation of SUMMARIZE-MINE, without further notices.

Figure 18.10 shows the corresponding number of patterns based on the same setting as Fig. 18.9, and we also add another curve depicting the fraction of false positives that is verified and discarded by the ID-list+top-down strategy. As expected, when min_sup' is reduced, false negatives decrease while false positives increase. The gap between these two curves corresponds to the number of subgraphs that are truly frequent in the original database D, which gradually widens as we move to the left of the picture, since SUMMARIZE-MINE can now catch more patterns above min_sup'. Accordingly, the price paid for this is an increased cost to mine the summarized database D' and verify against D.

We compare the performance of gSpan, a state-of-art graph miner, with SUMMARIZE-MINE in Fig. 18.11. For this experiment, a series of connected subgraphs are randomly drawn from each transaction, so that we can run both algorithms on graphs with different size and see whether there exists any trend. All other settings are the same as Fig. 18.9, except that we only run one iteration here.

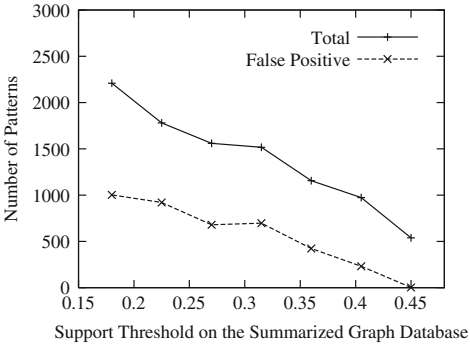


Fig. 18.10 Number of output patterns

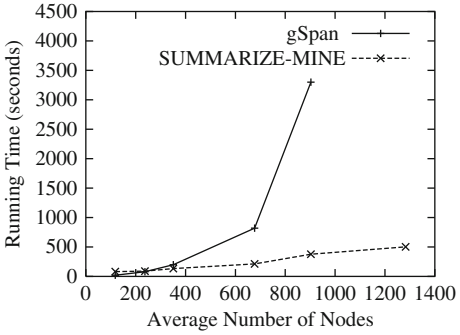


Fig. 18.11 Efficiency w.r.t. transaction size

Obviously, when the transaction size goes up, it becomes harder and harder for gSpan to work, where we have omitted the rightmost point of this curve since gSpan cannot finish within 3 h. In comparison, SUMMARIZE-MINE remains somewhat stable, which is natural, because the embedding enumeration issue becomes much worse for large graphs, and our algorithm is specifically designed to tackle this problem.

18.7.2 Synthetic Data Set

Generator Description. The synthetic graph generator follows a similar mechanism as the one used to generate itemset transactions, where we can set the number of graphs (D), average size of graphs (T), number of seed patterns (L), average size of seed patterns (I), and number of distinct vertex/edge labels (V/E). To begin with, a set of L seed patterns are generated randomly, whose size is determined by a Poisson distribution with mean I; then, seed patterns are randomly selected and inserted into a graph one by one until the graph reaches its size, which is the

realization of another Poisson variable with mean T. Due to lack of space, we refer interested readers to [15] for further details.

Figure 18.12 considers the problem of optimally setting the new support threshold min_sup' to achieve best algorithmic efficiency while ensuring a specific probabilistic guarantee, i.e., the overall false-negative rate is at most $\varepsilon = 0.05$. Considering the total running time, intuitively, with a low min_sup', we would miss fewer patterns in one round and thus may require a smaller number of iterations to reach the desired ε; however, it is also true that more time has to be spent in each round. So, what is the best trade-off? Since one-round miss rate as predicted by Corollary 1 is monotonically decreasing in $q(p)$, we can make the following statement. Focusing on a particular value of $q(p) = \theta$, if under this setting, we can guarantee that the overall false-negative rate $(1 - P)^t$ is at most ε; then for all patterns p' with $q(p') \geq \theta$, the probability for them to miss from the output must be less than ε, too. This θ value can be adjusted to tune SUMMARIZE-MINE accordingly toward larger/smaller patterns or patterns with more/less identically labeled vertices.

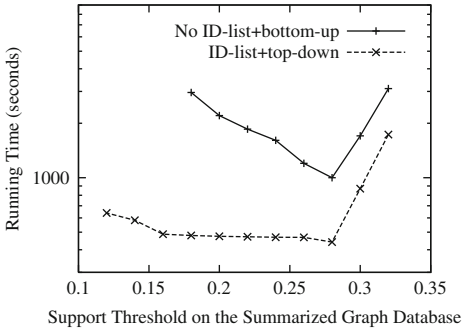


Fig. 18.12 The optimal setting of min_sup'

Setting $\theta = 0.8$ (which we think is reasonable for the mining task in hand), the total number of rounds t can be easily determined based on a given value of min_sup': Here, t is calculated by the formula $t = \lceil \frac{\log \varepsilon}{\log(1-P)} \rceil$, where $1 - P$ should be substituted by the probability bound given in Corollary 1. Running SUMMARIZE-MINE for t iterations, we can draw the total computation time against min_sup', which is shown in Fig. 18.12. The synthetic data set we take is D400T500L200I5V5E1, i.e., 400 transactions with 500 vertices on average, which are generated by 200 seed patterns of average size 5; the number of possible vertex/edge labels is set to 5/1. Considering the graphs we generated above, each transaction has approximately the same size, and thus it is reasonable to retain an equal number of $a = 50$ vertices for all summaries. min_sup is set to 40%. Finally, the lowest running time turns out to be reached at $min\_sup' = 28\%$ for both ID-list+top-down and No ID-list+bottom-up, where because of its ability to pre-prune at the very beginning, ID-list+top-down is not influenced much when min_sup' becomes low, which enables us to include more points for the corresponding curve

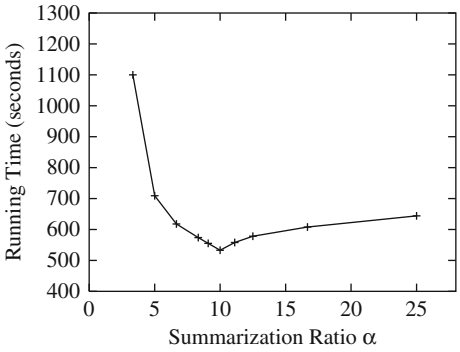


Fig. 18.13 Running time w.r.t. summarization ratio

when it is extended to the left. Also, the running time is not quite sensitive to parameter choices, as long as min_sup' is not too high.

In Fig. 18.13, we analyze the impact of summarization ratio on our algorithm. The data set is D500T500L200I5V5E1. We vary α from 3.33 to 25 (outer loop), while min_sup' is implicitly tuned to the best possible value as we did in Fig. 18.12 (inner loop). It can be seen that $\alpha = 10$ happens to be the optimal position: When we summarize more, data graphs become smaller, which makes it faster to mine frequent subgraphs over the summaries; however, in the meantime, topology collapsing also introduces more false negatives and false positives, where additional computing resources must be allocated to deal with them. In this sense, it is important to run SUMMARIZE-MINE at the best trade-off point; and as we can see from the figure, there are actually a broad range of summarization ratios with reasonable performance.

Taking D($|D|$)T500L200I5V5E1, we also test the efficiency of our algorithm over ten data sets by varying the number of transactions $|D|$ from 100, 200 up to 1,000, which is shown in Fig. 18.14. We use $min\_sup = 40\%$, $\alpha = 10$, while min_sup' and number of rounds t are tuned and optimally set as we did in

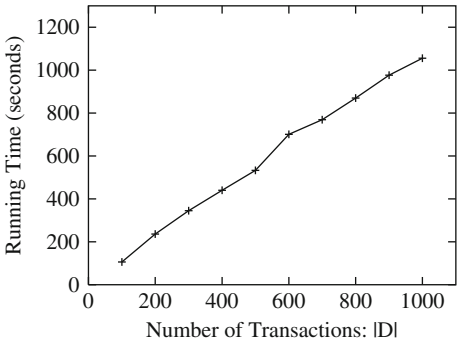


Fig. 18.14 Scalability test

Fig. 18.12. As demonstrated, the implementation is highly efficient, which can finish in hundreds of seconds, and SUMMARIZE-MINE is linearly scalable with regard to the data size.

18.8 Related Work

Many efficient frequent subgraph mining algorithms have been proposed, including FSG [15], gSpan [28], AGM [13], followed by Path-Join, MoFa, FFSM, GASTON, etc., and the wealth of literature cannot be fully enumerated here. Owing to more recent development, now we are also able to mine maximal graph patterns [12], significant graph patterns [10], and patterns with topological constraints [20]. All these methods directly take the input graph database without any data reduction. Their strategy works fine for a database consisting of small graphs, but could not perform efficiently when the graphs contain a large number of pattern embeddings, which we analyzed in the introduction.

There is another line of research [4, 16] that specifically mines frequent graph patterns from a single large network. Their major contribution is to define the pattern support in a single graph G, i.e., how many times should we count a pattern, given all its embeddings in G that might overlap? These methods are often restricted to sparse networks or networks with a good number of labels, thus limiting the number of potential embeddings.

There have been a few studies on how to improve the efficiency of graph mining in general. However, they approach the problem from different angles, and none of them could tackle the intrinsic difficulty of embedding enumeration in bulky graph data sets. To name a few, Yan et al. [27] introduce structural leap search and leverages structural proximity to mine discriminative subgraphs. Hasan et al. [9] invent a randomized heuristic to traverse the pattern space, where a collection of representative patterns are found. It analyzes how to reduce pattern candidates, based on the observation that many of them are quite similar. These two methods still work on the pattern space: Instead of doing a normal traversal, they can either perform "leap" or pick "delegates". To improve the mining speed on a large sparse graph, Rainhardt and Karypis [23] decide to incorporate parallel processing techniques, which are orthogonal to the focus of SUMMARIZE-MINE.

The concept of summarizing large graphs in order to facilitate processing and understanding is not new [11]. Raghavan and Garcia-Molina [22] study the problem of compressing Web graphs so that the link information can be efficiently stored and easily manipulated for fast computation of PageRank; Sarlos et al. [24] further analyze how the sketches can help calculate approximate personalized PageRank. Chakrabarti and Faloutsos [2] develop statistical summaries that analyze simple graph characteristics like degree distributions and hop-plots. Navlakha et al. [19] approximate a large network by condensing its nodes and edges, which can preserve the original topological skeleton within a bounded error. Recently, Tian et al. [25] suggest a semantics-oriented way to summarize graphs by grouping

vertices based on their associated attributes, which reflects the inherent structures and promotes easy user navigation; Chen et al. [3] further integrate this notion into a generic topological OLAP framework, where a graph cube can be built. The mining algorithm we developed in this paper can be further combined with all these studies to examine how structured patterns are presented on the summarized level.

Regarding other data reduction techniques that can be applied, we have pointed out sampling [26] and FP-Growth [8] as two examples that either reduce the number of transactions or compress between transactions, which are different from our compression method that takes effect within transactions. For a given pattern, because subgraph isomorphism checking and associated embedding enumerations happen inside a target graph, any method that cannot dig into individual transactions does not help. For instance, if we want to sample, then we should sample nodes/edges/substructures and in the meantime keep their original characteristics intact, so as to preserve the underlying patterns. This may require us to assume a generic graph generation model like the one given in [18]. In contrast, SUMMARIZE-MINE does not need such assumptions; the theoretical bound we developed is only conditional on the random grouping and merging of nodes, which can be easily implemented.

Finally, within a bigger context, the method of creating and leveraging synopsis to facilitate data processing has received significant attention in the broad database field [7, 30]. There is a recent work [17] on bursty sequence mining that transforms consecutive, identically labeled items within the same transaction into intervals for the purpose of length reduction. However, as the data becomes more complex and takes the form of graphs, compression based on randomized mechanisms plays a key role in pattern preserving, which is a major contribution of this study. For example, in XSKETCH [21], the same set of nodes in the XML graph are often merged together, which could cause much pattern loss if we perform mining on such kind of summaries.

18.9 Conclusions

In this chapter, we examine an important issue in frequent graph pattern mining, the intrinsic difficulty to perform embedding enumeration in large graphs, which might block many important downstream applications. Mining bulky graph data sets is in general very hard, but the problem should still be solvable if the node/edge labeling is not very diverse, which limits the explosion of pattern space. As we tried to find out the bottleneck, it was observed that even for small and simple substructures, the corresponding mining process could be very slow due to the existence of thousands of isomorphic embeddings in the target graphs. So, different from previous studies, SUMMARIZE-MINE proposes a novel mining framework that focuses on *data space reduction within transactions* and effectively turns lossy compression into a virtually lossless method by mining *randomized summaries* for multiple

iterations. Experimental results on real malware data demonstrate the efficiency of our method, which can find interesting malware fingerprints that were not revealed previously. Moreover, SUMMARIZE-MINE also sheds light on how data compression may impact the underlying patterns. This will be particularly interesting, given an emerging trend of huge information networks that must adopt data reduction as a necessary preprocessing step for analytical purposes.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.
2. D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Survey*, 38(1):1–69, 2006.
3. C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. Graph OLAP: Towards online analytical processing on graphs. In *ICDM*, pages 103–112, 2008.
4. J. Chen, W. Hsu, M.-L. Lee, and S.-K. Ng. Nemofinder: Dissecting genome-wide protein-protein interactions with meso-scale network motifs. In *KDD*, pages 106–115, 2006.
5. M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *ESEC/SIGSOFT FSE*, pages 5–14, 2007.
6. M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.
7. M. N. Garofalakis and P. B. Gibbons. Approximate query processing: Taming the terabytes (tutorial). In *VLDB*, 2001.
8. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD Conference*, pages 1–12, 2000.
9. M. A. Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki. Origami: Mining representative orthogonal graph patterns. In *ICDM*, pages 153–162, 2007.
10. H. He and A. K. Singh. Efficient algorithms for mining significant substructures in graphs with quality guarantees. In *ICDM*, pages 163–172, 2007.
11. L. B. Holder, D. J. Cook, and S. Djoko. Substucture discovery in the subdue system. In *KDD Workshop*, pages 169–180, 1994.
12. J. Huan, W. Wang, J. Prins, and J. Yang. Spin: Mining maximal frequent subgraphs from graph databases. In *KDD*, pages 581–586, 2004.
13. A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50(3):321–354, 2003.
14. S. Kramer, L. De Raedt, and C. Helma. Molecular feature mining in hiv data. In *KDD*, pages 136–143, 2001.
15. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001.
16. M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.
17. A. Lachmann and M. Riedewald. Finding relevant patterns in bursty sequences. *PVLDB*, 1(1):78–89, 2008.
18. J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.
19. S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD Conference*, pages 419–432, 2008.
20. J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005.

21. N. Polyzotis and M. N. Garofalakis. Xsketch synopses for xml data graphs. *ACM Transactions on Database Systems*, 31(3):1014–1063, 2006.
22. S. Raghavan and H. Garcia-Molina. Representing web graphs. In *ICDE*, pages 405–416, 2003.
23. S. Reinhardt and G. Karypis. A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph. In *IPDPS*, pages 1–8, 2007.
24. T. Sarlós, A. A. Benczúr, K. Csalogány, D. Fogaras, and B. Rácz. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *WWW*, pages 297–306, 2006.
25. Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *SIGMOD Conference*, pages 567–580, 2008.
26. H. Toivonen. Sampling large databases for association rules. In *VLDB*, pages 134–145, 1996.
27. X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. In *SIGMOD Conference*, pages 433–444, 2008.
28. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.
29. X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD Conference*, pages 335–346, 2004.
30. N. Zhang, V. Kacholia, and M. T. Özsu. A succinct physical storage scheme for efficient evaluation of path queries in xml. In *ICDE*, pages 54–65, 2004.

Part VI
Analysis of Biological Information Networks

Chapter 19
Finding High-Order Correlations in High-Dimensional Biological Data

Xiang Zhang, Feng Pan, and Wei Wang

Abstract In many emerging real-life problems, the number of dimensions in the data sets can be from thousands to millions. The large number of features poses great challenge to existing high-dimensional data analysis methods. One particular issue is that the latent patterns may only exist in subspaces of the full-dimensional space. In this chapter, we discuss the problem of finding correlations hidden in feature subspaces. Both linear and nonlinear cases will be discussed. We present efficient algorithms for finding such correlated feature subsets.

19.1 Introduction

Many real-life applications involve the analysis of high-dimensional data. For example, in bio-medical domains, advanced microarray techniques [1, 2] enable monitoring the expression levels of hundreds to thousands of genes simultaneously. By mapping each gene to a feature, gene expression data can be represented by points in a high-dimensional feature space. To make sense of such high-dimensional data, extensive research has been done in finding the latent structure among the large number of features. In general, existing approaches in analyzing high-dimensional data can be summarized into three categories [3]: feature selection, feature transformation (or dimension reduction), and projected clustering.

The goal of feature selection methods [4–7] is to find a single representative subset of features that are most relevant for the task at hand, such as classification. The selected features generally have low correlation with each other but have strong correlation with the target feature.

Feature transformation methods [8–13] summarize the data set by creating linear/nonlinear combinations of features in order to uncover the latent structure. The insight behind feature transformation methods is that a high dimensional data set may exhibit interesting patterns on a lower dimensional subspace

W. Wang (✉)
Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA
e-mail: weiwang@cs.unc.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_19,

due to correlations among the features. The commonly used linear feature transformation methods include principal component analysis (PCA) [11], linear discriminant analysis (LDA), and their variants (see [9] for an overview). PCA is one of the most widely used feature transformation methods. It seeks an optimal linear transformation of the original feature space such that most variance in the data is represented by a small number of orthogonal derived features in the transformed space. PCA performs one and the same feature transformation on the entire data set. It aims to model the global latent structure of the data and hence does not separate the impact of any original features nor identify local latent patterns in some feature subspaces.

Recently proposed projected clustering methods, such as [14, 15], can be viewed as combinations of clustering algorithms and PCA. These methods can be applied to find clusters of data points that may not exist in the axis parallel subspaces but only exist in the projected subspaces. The projected subspaces are usually found by applying the standard PCA in the full dimensional space. Like other clustering methods, projected clustering algorithms find the clusters of points that are spatially close to each other in the projected space. However, a subset of features can be strongly correlated even though the data points do not form any clustering structure.

19.1.1 Motivation

In many emerging applications, the data sets usually consist of thousands to hundreds of thousands of features. In such high-dimensional data set, some feature subsets may be strongly correlated, while others may not have any correlation at all. In these applications, it is more desirable to find the correlations that are hidden in feature subspaces. For example, in gene expression data analysis, a group of genes having strong correlation is of high interest to biologists since it helps to infer unknown functions of genes [1] and gives rise to hypotheses regarding the mechanism of the transcriptional regulatory network [2]. We refer to such correlation among a subset of features as a *local correlation* in comparison with the global correlation found by the full dimensional feature reduction methods. Since such local correlations only exist in some subspaces of the full dimensional space, they are invisible to the full feature transformation methods.

Recently, many methods [1, 16] have been proposed for finding clusters of features that are pairwisely correlated. However, a set of features may have strong correlation but each pair of features only weakly correlated.

For example, Fig. 19.1 shows four genes that are strongly correlated in the mouse gene expression data collected by the biologists in the School of Public Health at UNC. All of these four genes have same Gene Ontology (GO) [17] annotation *cell part*, and three of which, Myh7, Hist1h2bk, and Arntl, share the same GO annotation *intracelluar part*. The linear relationship identified by our algorithm is $-0.4(Nrg4) + 0.1(Myh7) + 0.7(Hist1h2bk) - 0.5(Arntl) = 0$. As we can see from the figure, all data points almost perfectly lay on the same hyperplane, which shows that the four genes are strong correlated. (In order to visualize this three-dimensional hyperplane, we combine two features, Nrg4 and Myh7, into a single

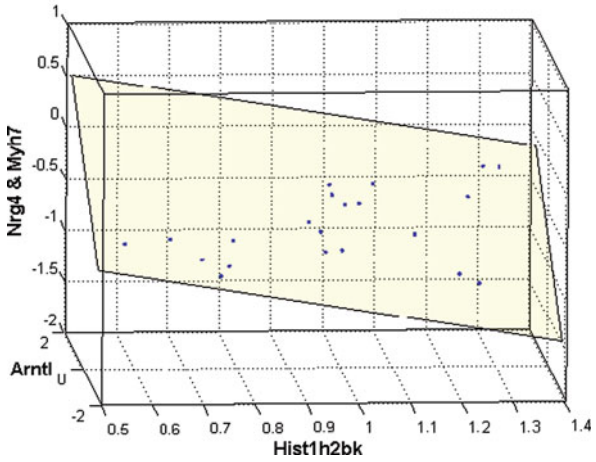


Fig. 19.1 A strongly correlated gene subset

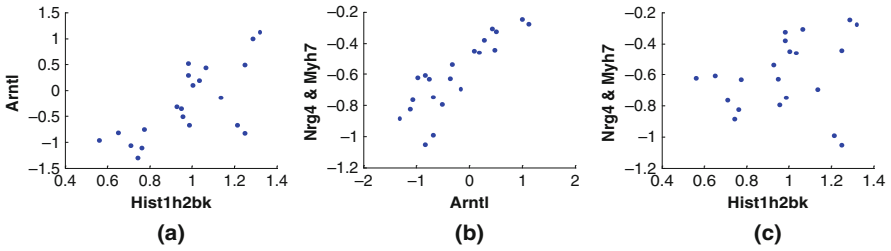


Fig. 19.2 Pairwise correlations of a strongly correlated gene subset. (**a**) (Hist1h2bk, Arntl); (**b**) (Arntl, Nrg4&Myh7); (**c**) (Hist1h2bk, Nrg4&Myh7)

axis as $-0.4(Nrg4) + 0.1(Myh7)$ to reduce it to a two-dimensional hyperplane.) If we project the hyperplane onto two dimensional spaces formed by each pair of genes, we find none of them show strong correlation, as depicted in Fig. 19.2a–c.

Projected clustering algorithms [14] have been proposed to find the clusters of data points in projected feature spaces. This is driven by the observation that clusters may exist in arbitrarily oriented subspaces. Like other clustering methods, these methods tend to find the clusters of points that are spatially close to each other in the feature space. However, as shown in Fig. 19.1, a subset of features (genes in this example) can still be strongly correlated even if the data points are far away from each other. This property makes such strong correlations invisible to the projected clustering methods. Moreover, to find the projections of original features, projected clustering methods apply PCA in the full dimensional space. Therefore, they cannot decouple the local correlations hidden in the high-dimensional data.

In [18], an algorithm is proposed to find local linear correlations in high-dimensional data. However, in real applications, the feature subspace can be either linearly or nonlinearly correlated. The problem of finding linear and nonlinear correlations in feature subspacesremains open.

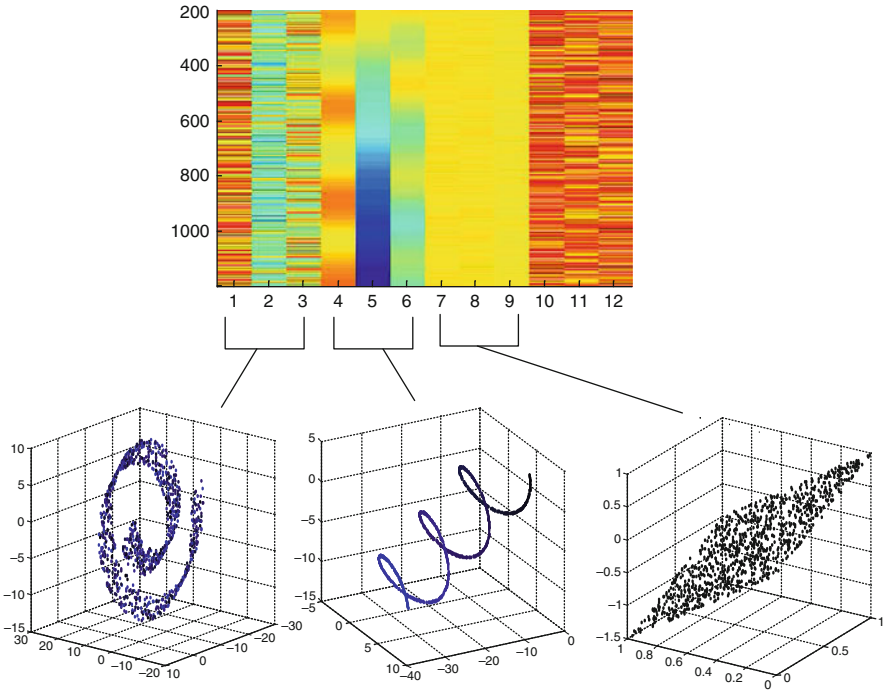


Fig. 19.3 An example data set

For example, Fig. 19.3 shows a data set consisting of 12 features, $\{f_1, f_2, \ldots, f_{12}\}$ and 1000 data points. Embedded in the full dimensional space, features subspaces $\{f_1, f_2, f_3\}$ and $\{f_4, f_5, f_6\}$ are nonlinearly correlated and $\{f_7, f_8, f_9\}$ are linearly correlated. Features $\{f_{10}, f_{11}, f_{12}\}$ contain random noises.

Performing feature transformation methods to the full dimensional space cannot uncover these local correlations hidden in the full feature spaces. For example, Fig. 19.4a shows the result of applying Principal Component Analysis (PCA) [11] to the full dimensional space of the example data set shown in Fig. 19.3. In this figure, we plot the point distribution on the first three principal components found by PCA. Clearly, we cannot find any pattern that is similar to the patterns embedded in the data set. Similarly, Fig. 19.4b shows the results of applying ISOMAP [13] to reduce the dimensionality of the data set down to 3. There is also no desired pattern found in this low-dimensional structure.

How can we identify these local correlations hidden in the full dimensional space?

This question is twofold. First, we need to identify the strongly correlated feature subspaces, i.e., a subset of features that are strongly correlated and actually have low-dimensional structures. Then, after these locally correlated feature subsets are found, we can apply the existing dimensionality reduction methods to identify the low-dimensional structures embedded in them.

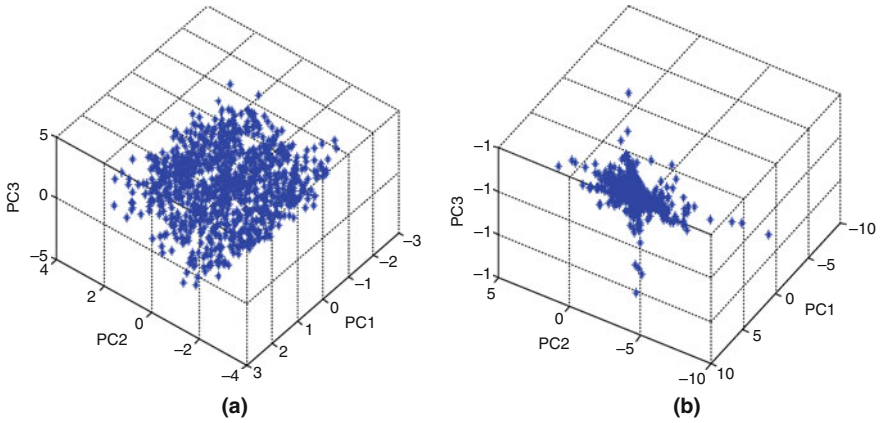


Fig. 19.4 Applying dimensionality reduction methods to the full dimensional space of the example data set. (**a**) Result of PCA (**b**) Result of ISOMAP

Many methods have been proposed to address the second aspect of the question, i.e., given a correlated feature space, finding the low-dimensional embedding in it. The first aspect of the question, however, is largely untouched. In this chapter, we investigate the first aspect of the question, i.e., identifying the strongly correlated feature subspaces.

19.1.2 Challenges and Contributions

(1) In this chapter, we investigate the problem of finding correlations hidden in the feature subspaces of high-dimensional data. The correlations can be either linear or nonlinear. To our best knowledge, our work is the first attempt to find local linear and nonlinear correlations hidden in feature subspaces.

For both linear and nonlinear cases, we formalize the problem as finding *reducible subspaces* in the full dimensional space. We adopt the concept of PCA [11] to model the linear correlations. The PCA analysis is repeated applied on subsets of features. In the nonlinear cases, we utilize intrinsic dimensionality [19] to detect reducible subspaces. Informally, a set of features are correlated if the intrinsic dimensionality of the set is smaller than the number of features. Various intrinsic dimensionality estimators have been developed [20–22]. Our problem formalization does not depend on any particular method for estimating the intrinsic dimensionality.

(2) We develop an efficient algorithm, CARE,[1] for finding local linear correlations. CARE utilizes spectrum properties about the eigenvalues of the covariance matrix and incorporates effective heuristic to improve the efficiency.

(3) We develop an effective algorithm REDUS[2] to detect nonlinearly correlated feature subsets. REDUS consists of the following two steps.

[1] CARE stands for finding loCAl lineaR corrElations.

[2] REDUS stands for REDUcible Subspaces.

It first finds the union of all reducible subspaces, i.e., the *overall reducible subspace*. The second step is to uncover the individual reducible subspaces in the overall reducible subspace. The key component of this step is to examine if a feature is strongly correlated with a feature subspace. We develop a method utilizing point distributions to distinguish the features that are strongly correlated with a feature subspace and those that are not. Our method achieves similar accuracy to that of directly using intrinsic dimensionality estimators, but with much less computational cost.

Extensive experiments on synthetic and real-life data sets demonstrate the effectiveness of CARE and REDUS.

19.2 Related Work

19.2.1 Feature Transformation

Feature transformation methods can be categorized into linear methods, such as multi-dimensional scaling (MDS) [10] and principal component analysis (PCA) [11], and nonlinear methods, such as local linear embedding (LLE) [12], ISOMAP [13], and Laplacian eigenmaps [8]. For high dimensional data sets, if there exist low-dimensional subspaces or manifolds embedded in the full dimensional spaces, these methods are successful in identifying these low-dimensional embeddings.

Feature transformation methods are usually applied on the full dimensional space to capture the independent components among all the features. They are not designed to address the problem of identifying correlation in feature subspaces. It is reasonable to apply them to the feature spaces that are indeed correlated. However, in very high dimensional data sets, different feature subspaces may have different correlations, and some feature subspace may not have any correlation at all. In this case, dimensionality reduction methods should be applied after such strongly correlated feature subspaces have been identified.

19.2.2 Feature Selection

Feature selection methods [4–7] try to find a subset of features that are most relevant for certain data mining task, such as classification. The selected feature subset usually contains the features that have low correlation with each other but have strong correlation with the target feature. In order to find the relevant feature subset, these methods search through various subsets of features and evaluate these subsets according to certain criteria. Feature selection methods can be further divided into two groups based on their evaluation criteria: wrapper and filter. Wrapper models evaluate feature subsets by their predictive accuracy using statistical re-sampling or cross-validation. In filter techniques, the feature subsets are evaluated by their information content, typically statistical dependence or information-theoretic measures.

Similar to feature transformation, feature selection finds one feature subset for the entire data set.

19.2.3 Subspace Clustering

Subspace clustering is based on the observation that clusters of points may exist in different subspaces. Many methods [23–25] have been developed to find clusters in axes paralleling subspaces. Recently, the projected clustering was studied in [14], inspired by the observation that clusters may exist in arbitrarily oriented subspaces. These methods can be treated as combinations of clustering algorithms and PCA. Similar to other clustering methods, these methods tend to find the clusters of points that are close to each other in the projected space. However, as shown in Fig. 19.1, a subset of features still can be strongly correlated even if the data points are far away from each other. Pattern-based bi-clustering algorithms have been studied in [1, 16]. These algorithms find the clusters in which the data points share pairwise linear correlations, which is only a special case of linear correlation.

19.2.4 Intrinsic Dimensionality

Due to correlations among features, a high-dimensional data set may lie in a subspace with dimensionality smaller than the number of features [20–22]. The intrinsic dimensionality can be treated as the minimum number of free variables required to define the data without any significant information loss [19]. For example, as shown in Fig. 19.3, in the three-dimensional space of $\{f_1, f_2, f_3\}$, the data points lie on a Swiss roll, which is actually a two-dimensional manifold. Therefore, its intrinsic dimensionality is 2.

The concept of intrinsic dimensionality has many applications in the database and data mining communities, such as clustering [26, 27], outlier detection [28], nearest neighbor queries [29], and spatial query selectivity estimation [30, 31]. Different definitions of intrinsic dimensionality can be found in the literature. For example, in linear cases, matrix rank [32] and PCA [11] can be used to estimate intrinsic dimensionality. For nonlinear cases, estimators such as *box counting dimension*, *information dimension*, and *correlation dimension* have been developed. These intrinsic dimensionality estimators are sometimes collectively referred to as *fractal dimension*. Please see [33, 34] for good coverage of the topics of intrinsic dimensionality estimation and its applications.

19.3 Problem Formalization

In this section, we utilize PCA and intrinsic dimensionality to formalize the problem of finding strongly correlated feature subspaces in linear and nonlinear cases, respectively .

Suppose that the data set Ω consists of N data points and M features. Let $\Omega_P = \{p_1, p_2, \ldots, p_N\}$ denote the point set, and $\Omega_F = \{f_1, f_2, \ldots, f_M\}$ denote the feature set in Ω, respectively. In the following sections, we define the linear and nonlinear reducible subspaces.

19.3.1 Linear Reducible Subspace

A strongly linear-correlated feature subset is a subset of features that show strong linear correlation in a large portion of data points.

Definition 1 (STRONGLY LINEAR-CORRELATED FEATURE SUBSET)
Let $\Omega' = \{\mathbf{f}_{i_1}, \ldots, \mathbf{f}_{i_m}\} \times \{\mathbf{p}_{j_1}, \ldots, \mathbf{p}_{j_n}\}$ be a submatrix of Ω, where $1 \leq i_1 < i_2 < \cdots < i_m \leq M$ and $1 \leq j_1 < j_2 < \ldots < j_n \leq N$. C_F is the covariance matrix of Ω'. Let $\{\lambda_l\}$ $(1 \leq l \leq n)$ be the eigenvalues of C_F and arranged in increasing order,[3] i.e., $\lambda_1 \leq \lambda_2, \cdots, \leq \lambda_n$. The features $\{\mathbf{f}_{i_1}, \ldots, \mathbf{f}_{i_m}\}$ is a *strongly linear-correlated feature subset* if the value of the objective function $f(\Omega', k) = \frac{\Sigma_{t=1}^{k} \lambda_t}{\Sigma_{t=1}^{m} \lambda_t} \leq \eta$ and $n/N \geq \delta$, where k, η, and δ are user-specified parameters.

Eigenvalue λ_l is the variance on eigenvector $\mathbf{v}_l$ [11]. The set of eigenvalues $\{\lambda_l\}$ of matrix $C_{\Omega'}$ is also called the *spectrum* of $C_{\Omega'}$ [35].

Geometrically, each $n \times m$ submatrix of Ω represents an m-dimensional space with n points in it. This m-dimensional space can be partitioned into two subspaces, S_1 and S_2, which are orthogonal to each other. S_1 is spanned by the k eigenvectors with smallest eigenvalues and S_2 is spanned by the remaining $m - k$ eigenvectors. Intuitively, if the variance in subspace S_1 is small (equivalently the variance in S_2 is large), then the feature subset is strongly linear-correlated. The input parameters k and threshold η for the objective function $f(\Omega', k) = \frac{\Sigma_{t=1}^{k} \lambda_t}{\Sigma_{t=1}^{m} \lambda_t}$ are used to control the strength of the correlation among the feature subset. The default value of k is 1. The larger the value of k, the stronger the linear correlation.

The reason for requiring $n/N \geq \delta$ is because a feature subset can be strongly linear-correlated only in a subset of data points. In our definition, we allow the strongly linear-correlated feature subsets to exist in a large portion of the data points in order to handle this situation. Note that it is possible that a data point may participate in multiple local correlations held by different feature subsets. This makes the local correlations more difficult to detect. Please also note that for a given strongly linear-correlated feature subset, it is possible that there exist multiple linear correlations on different subsets of points. In this chapter, we focus on the scenario where there exists only one linear correlation for a strongly linear-correlated feature subset.

[3] In this chapter, we assume that the eigenvalues are always arranged in increasing order. Their corresponding eigenvectors are $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
p_1	0.7988	**3.8905**	0.6548	−0.6646	0.5536	1.3242	**−0.5532**	0.3158	**−1.1613**
p_2	0.8968	**1.3365**	−1.2484	0.5582	−1.5564	−0.1265	**0.2983**	1.3437	**−1.1098**
p_3	0.1379	**3.1503**	−0.5975	−1.1885	−0.2067	−0.7372	**−1.2266**	−2.2378	**0.2907**
p_4	−1.6191	**3.9939**	−0.4818	−0.7755	−0.4256	0.2137	**−0.1897**	1.2929	**−1.9102**
p_5	−1.6466	**−1.1069**	0.9834	0.271	0.4938	−0.4005	**−0.3017**	−0.3785	**1.3148**
p_6	0.4287	**−4.0447**	1.7621	1.535	−0.8709	0.0649	**0.957**	0.0025	**0.6653**
p_7	**−0.982**	**2.1536**	1.4274	−1.0523	**0.0798**	**−1.758**	**−0.5334**	**0.8846**	**−0.2751**
p_8	**−5.1084**	**2.7252**	0.9118	0.6256	**−0.5216**	**1.6867**	**−0.9011**	**0.5825**	**−0.023**
p_9	**10.9007**	**4.3305**	0.3268	−0.7976	**−1.4139**	**0.3274**	**−0.8926**	**−1.6142**	**−0.908**
p_{10}	**7.3744**	−0.8533	0.0696	−0.3135	**−0.3843**	**0.716**	0.2787	**−1.5037**	−1.0437
p_{11}	**−4.9437**	0.3456	−1.4998	−0.6022	**−0.4579**	**1.5986**	−0.7458	**0.5736**	0.3735
p_{12}	**9.7836**	0.1098	−0.4182	1.2591	**−0.2915**	**−2.0647**	1.6035	**−0.9105**	0.9015
p_{13}	**10.9429**	−1.133	−0.021	0.8585	**−0.3012**	**−0.7436**	0.5743	**−1.6313**	1.2785
p_{14}	**5.9643**	−0.6831	0.2284	−2.1053	**−1.5886**	**0.1762**	0.3207	**−0.3591**	−0.1285
p_{15}	**−2.0985**	−0.2779	−1.0082	−0.3609	**1.0943**	**0.5278**	−0.1514	**−0.3976**	0.6128

Fig. 19.5 An example data set containing linear-correlated feature subsets

For example, in the data set shown in Fig. 19.5, the features in submatrix $\Omega' = \{\mathbf{f}_2, \mathbf{f}_7, \mathbf{f}_9\} \times \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_9\}$ is a strongly linear-correlated feature subset when $k = 1$, $\eta = 0.004$, and $\delta = 60\%$. The eigenvalues of the covariance matrix, $C_{\Omega'}$, the input parameters, and the value of the objective function are shown in Table 19.1.

The spectrum of covariance matrix has a well-known theorem which is often called the *interlacing eigenvalues theorem*[4] [35].

Table 19.1 An example of strongly linear-correlated feature subset

Feature subset	$\{\mathbf{f}_2, \mathbf{f}_7, \mathbf{f}_9\}$
Eigenvalues of $C_{\Omega'}$	$\lambda_1 = 0.001, \lambda_2 = 0.931, \lambda_3 = 2.067$
Input parameters	$k = 1$, $\eta = 0.004$ and $\delta = 60\%$
Objective function value	$f(\Omega', k) = 0.0003$

[4] This theorem also applies to Hermitian matrix [35]. Here we focus on the covariance matrix, which is semi-positive definite and symmetric.

Theorem 1 *Let* $\Omega' = \{\mathbf{f}_{i_1}, \ldots, \mathbf{f}_{i_m}\} \times \{\mathbf{p}_{j_1}, \ldots, \mathbf{p}_{j_n}\}$ *and* $\Omega'' = \{\mathbf{f}_{i_1}, \ldots, \mathbf{f}_{i_m}, \mathbf{f}_{i_{(m+1)}}\} \times \{\mathbf{p}_{j_1}, \ldots, \mathbf{p}_{j_n}\}$ *be two submatrices of* Ω. $C_{Omega'}$ *and* $C_{\Omega''}$ *are their covariance matrices with eigenvalues* $\{\lambda_l\}$ *and* $\{\lambda_l^{'}\}$. *We have*

$$\lambda_1^{'} \leq \lambda_1 \leq \lambda_2^{'} \leq \lambda_2 \leq \cdots \leq \lambda_{m-1} \leq \lambda_m^{'} \leq \lambda_m \leq \lambda_{m+1}^{'}.$$

Theorem 1 tells us that the spectra of $C_{\Omega'}$ and $C_{\Omega''}$ interleave each other, with the eigenvalues of the larger matrix bracketing those of the smaller one.

By applying the interlacing eigenvalues theorem, we have the following property for the strongly linear-correlated feature subsets.

Property 1 (Upward closure property of strongly linear-correlated feature subsets) Let $\Omega' = V' \times P$ and $\Omega'' = V'' \times P$ be two submatrices of Ω with $V' \subseteq V''$. If V' is a strongly linear-correlated feature subset, then V'' is also a strongly linear-correlated feature subset.

Proof We show the proof for the case where $|V''| = |V'| + 1$, i.e., V' is a subset of V'' by deleting one feature from V'. Let $C_{\Omega'}$ and $C_{\Omega''}$ be the covariance matrices of Ω' and Ω'' with eigenvalues $\{\lambda_l\}$ and $\{\lambda_l^{'}\}$. Since V' is a strongly linear-correlated feature subset, we have $f(\Omega', k) = \dfrac{\Sigma_{t=1}^{k} \lambda_t}{\Sigma_{t=1}^{m} \lambda_t} \leq \eta$. By applying the interlacing eigenvalues theorem, we have $\Sigma_{t=1}^{k} \lambda_t \geq \Sigma_{t=1}^{k} \lambda_t^{'}$ and $\Sigma_{t=1}^{m} \lambda_t \leq \Sigma_{t=1}^{m+1} \lambda_t^{'}$. Thus $f(\Omega'', k) = \dfrac{\Sigma_{t=1}^{k} \lambda_t^{'}}{\Sigma_{t=1}^{m+1} \lambda_t^{'}} \leq \eta$. Therefore, V'' is also a strongly linear-correlated feature subset. By induction we can prove for the cases where V' is a subset of V'' by deleting more than one feature.

The following example shows the monotonicity of the objective function with respect to the feature subsets. Using the data set shown in Fig. 19.5, let $\Omega_1 = V_1 \times P_1 = \{\mathbf{f}_2, \mathbf{f}_7\} \times \{\mathbf{p}_1, \ldots, \mathbf{p}_{15}\}$, $\Omega_1' = (V_1 \cup \{\mathbf{f}_9\}) \times P_1$, and $\Omega_1'' = (V_1 \cup \{\mathbf{f}_4, \mathbf{f}_9\}) \times P_1$. The values of the objective function, when $k = 1$, are shown in Table 19.2. It can be seen from the table that the value of the objective function monotonically decreases when adding new features.

Table 19.2 Monotonicity with respect to feature subsets

Point subset $P_1 = \{\mathbf{p}_1, \ldots, \mathbf{p}_{15}\}$	
Feature subset V_1	$f(\Omega_1, k) = 0.1698$
Feature subset $V_1 \cup \{\mathbf{f}_9\}$	$f(\Omega_1', k) = 0.0707$
Feature subset $V_1 \cup \{\mathbf{f}_4, \mathbf{f}_9\}$	$f(\Omega_1'', k) = 0.0463$

On the other hand, adding (or deleting) data points to a fixed feature subset may cause the correlation of the features to either increase or decrease; that is, the objective function is non-monotonic with respect to the point subsets. We use the following example to show the non-monotonicity of the objective function with respect to the point subsets. Using the data set shown in Fig. 19.5, let

$\Omega_2 = V_2 \times P_2 = \{\mathbf{f}_2, \mathbf{f}_7, \mathbf{f}_9\} \times \{\mathbf{p}_1, \cdots, \mathbf{p}_9, \mathbf{p}_{11}\}$, $\Omega_2' = V_2 \times (P_2 \cup \{\mathbf{p}_{10}\})$, and $\Omega_2'' = V_2 \times (P_2 \cup \{\mathbf{p}_{14}\})$. The values of their objective functions, when $k = 1$, are shown in Table 19.3. It can be seen from the table that the value of the objective function f can either increase or decrease when adding more points.

Table 19.3 No monotonicity with respect to point subsets

Feature subset $V_2 = \{\mathbf{f}_2, \mathbf{f}_7, \mathbf{f}_9\}$	
Point subset P_2	$f(\Omega_2, k) = 0.0041$
Point subset $P_2 \cup \{\mathbf{p}_{10}\}$	$f(\Omega_2', k) = 0.0111$
Point subset $P_2 \cup \{\mathbf{p}_{14}\}$	$f(\Omega_2'', k) = 0.0038$

We define the linear reducible subspace.

Definition 2 (LINEAR REDUCIBLE SUBSPACE)
A submatrix $\Omega' = V \times P$ is a linear reducible subspace *iff* (1) Feature set V is strongly linear-correlated; (2) none of the feature subsets of V is strongly linear-correlated.

19.3.2 Nonlinear Reducible Subspace

PCA can only measure linear correlations. In this section, we extend the problem to nonlinear correlations and nonlinear reducible subspaces. Instead of specifically using "nonlinear," we use the general terms "correlation" and "reducible subspace", for both linear and nonlinear cases.

We use intrinsic dimensionality to define correlated features (linear and nonlinear). Given a submatrix $\Omega' = V \times P$, we use $\mathrm{ID}(V)$ to represent the intrinsic dimensionality of the feature subspace $V \in \Omega_F$. Intrinsic dimensionality provides a natural way to examine whether a feature is correlated with some feature subspace: if a feature $f_a \in \Omega_F$ is strongly correlated with a feature subspace $V \subseteq \Omega_F$, then adding f_a to V should not cause much change of the intrinsic dimensionality of V. The following definition formalizes this intuition.

Definition 3 (STRONG CORRELATION)
A feature subspace $V \subseteq \Omega_F$ and a feature $f_a \in \Omega_F$ have strong correlation if

$$\Delta\mathrm{ID}(V, f_a) = \mathrm{ID}(V \cup \{f_a\}) - \mathrm{ID}(V) \leq \epsilon.$$

In this definition, ϵ is a user-specified threshold. Smaller ϵ value implies stronger correlation, and larger ϵ value implies weaker correlation. If V and f_a have strong correlation, we also say that they are strongly correlated.

Definition 4 (REDUNDANCY)
Let $V = \{f_{v_1}, f_{v_2}, \ldots, f_{v_m}\} \subseteq \Omega_F$. $f_{v_i} \in V$ is a redundant feature of V if f_{v_i} has strong correlation with the feature subspace consisting of the remaining features of V, i.e.,

$$\Delta\mathrm{ID}\left(\left\{f_{v_1}, \ldots, f_{v_{i-1}}, f_{v_{i+1}}, \ldots, f_{v_m}\right\}, f_{v_i}\right) \leq \epsilon.$$

We say V is a *redundant* feature subspace if it has at least one redundant feature. Otherwise, V is a *non-redundant* feature subspace.

Note that in Definitions 3 and 4, $ID(V)$ does not depend on a particular intrinsic dimensionality estimator. Any existing estimator can be applied when calculating $ID(V)$. Moreover, we do not require that the intrinsic dimensionality estimator reflects the exact dimensionality of the data set. However, in general, a good intrinsic dimensionality estimator should satisfy two basic properties.

First, if a feature is redundant in some feature subspace, then it is also redundant in the supersets of the feature subspace. We formalize this intuition as the following property.

Property 2 For $V \in \Omega_F$, if $\Delta\mathrm{ID}(V, f_a) \leq \epsilon$, then $\forall U\ (V \subseteq U \subseteq \Omega_F)$, $\Delta\mathrm{ID}(U, f_a) \leq \epsilon$.

This is a reasonable requirement, since if f_a is strongly correlated with $V \subseteq U$, then adding f_a to U will not greatly alter its intrinsic dimensionality.

From this property, it is easy to see that if feature subspace U is non-redundant, then all of its subsets are non-redundant, which is clearly a desirable property for the feature subspaces.

Corollary 1 *If $U \subseteq \Omega_F$ is non-redundant, then for $\forall V \subseteq U$, V is also non-redundant.*

The following property extends the concept of basis [36] in a linear space to nonlinear space using intrinsic dimensionality. In linear space, suppose that V and U contain the same number of vectors, and the vectors in V and U are all linearly independent. If the vectors of U are in the subspace spanned by the vectors of V, then the vectors in V and the vectors in U span the same subspace. (A span of a set of vectors consists of all linear combinations of the vectors.) Similarly, in Property 3, for two non-redundant feature subspaces, V and U, we require that if the features in U are strongly correlated with V, then U and V are strongly correlated with the same subset of features.

Property 3 Let $V = \{f_{v_1}, f_{v_2}, \ldots, f_{v_m}\} \subseteq \Omega_F$ and $U = \{f_{u_1}, f_{u_2}, \ldots, f_{u_m}\} \subseteq \Omega_F$ be two non-redundant feature subspaces. If $\forall f_{u_i} \in U$, $\Delta\mathrm{ID}(V, f_{u_i}) \leq \epsilon$, then for $\forall f_a \in \Omega_F$, $\Delta\mathrm{ID}(U, f_a) \leq \epsilon$ iff $\Delta\mathrm{ID}(V, f_a) \leq \epsilon$.

Intuitively, if a feature subspace Y ($Y \subseteq \Omega_F$) is redundant, then Y should be reducible to some subspace, say V ($V \subset Y$). Concerning the possible choices of V, we are most interested in the smallest one that Y can be reduced to, since it represents the intrinsic dimensionality of Y. We now give the formal definitions of reducible subspace and its core space.

Definition 5 (Reducible Subspace and Core Space)
$Y \subseteq \Omega_F$ is a reducible subspace (linear or nonlinear) if there exists a non-redundant subspace V ($V \subset Y$), such that

(1) $\forall f_a \in Y$, $\Delta\text{ID}(V, f_a) \leq \epsilon$ and
(2) $\forall U \subset Y$ ($|U| \leq |V|$), U is non-redundant.

We say V is the core space of Y, and Y is reducible to V.

Criterion (1) in Definition 5 says that all features in Y are strongly correlated with the core space V. The meaning of criterion (2) is that the core space is the smallest non-redundant subspace of Y with which all other features of Y are strongly correlated.

Among all reducible subspaces, we are most interested in the maximum ones. A maximum reducible subspace is a reducible subspace that includes all features that are strongly correlated with its core space.

Definition 6 (MAXIMUM REDUCIBLE SUBSPACE)
$Y \subseteq \Omega_F$ is a maximum reducible subspace if

(1) Y is a reducible subspace and
(2) $\forall f_b \in \Omega_F$, if $f_b \notin Y$, then $\Delta\text{ID}(V, f_b) > \epsilon$, where V is the core space of Y.

Let $\{Y_1, Y_2, \ldots, Y_S\}$ be the set of all maximum reducible subspaces in the data set. The union of the maximum reducible subspaces $OR = \bigcup_{i=1}^{S} Y_i$ is referred to as the *overall reducible subspace*.

Note that Definition 6 works for the general case where a feature can be in different maximum reducible subspaces. In this chapter, we focus on the special case where maximum reducible subspaces are non-overlapping, i.e., each feature can be in *at most one* maximum reducible feature subspace.

In the following sections, we present the CARE and REDUS algorithms which efficiently detect linear reducible subspaces and maximum (nonlinear) reducible subspaces in high-dimensional data.

19.4 The CARE Algorithm

In this section, we present the algorithm CARE for finding the linear reducible subspace (Definition 2). CARE enumerates the combinations of features to generate candidate feature subsets. To examine if a candidate is a linear reducible subspace, CARE adopts a two-step approach. It first checks if the feature subset is strongly correlated on all data points. If not, CARE then apply point deletion heuristic to find the appropriate subset of points on which the current feature subset may become strongly correlated. In Section 19.4.1, we first discuss the overall procedure of enumerating candidate feature subsets. In Section 19.4.2, we present the heuristics for choosing the point subsets for the candidates that are not strongly correlated on all data points.

19.4.1 Feature Subsets Selection

For any submatrix $\Omega' = V \times \{\mathbf{p}_1, \ldots, \mathbf{p}_M\}$ of Ω, in order to check whether feature subset V' is strongly correlated, we can perform PCA on Ω' to see if its objective function value is lower than the threshold, i.e., if $f(\Omega', k) = \frac{\Sigma_{t=1}^{k} \lambda_t}{\Sigma_{t=1}^{m} \lambda_t} \leq \eta$.

Starting from feature subsets containing a single feature, CARE adopts depth-first search to enumerate combinations of features to generate candidate feature subsets. In the enumeration process, if we find that a candidate feature subset is strongly correlated by evaluating its objective function value, then all its supersets can be pruned according to Property 1.

Next, we present an upper bound on the value of the objective function, which can help to speed up the evaluation process. The following theorem shows the relationship between the diagonal entries of a covariance matrix and its eigenvalues [35].

Property 4 Let Ω' be a submatrix of Ω and $C_{\Omega'}$ be the $m \times m$ covariance matrix of Ω'. Let $\{a_i\}$ be the diagonal entries of $C_{\Omega'}$ arranged in increasing order, and $\{\lambda_i\}$ be the eigenvalues of $C_{\Omega'}$ arranged in increasing order. Then $\Sigma_{t=1}^{s} a_t \geq \Sigma_{t=1}^{s} \lambda_t$ for all $s = 1, 2, \ldots, n$, with equality held for $s = m$.

Applying Property 4, we can get the following proposition.

Proposition 1 *Let Ω' be a submatrix of Ω and $C_{\Omega'}$ be the $m \times m$ covariance matrix of Ω'. Let $\{a_i\}$ be the diagonal entries of $C_{\Omega'}$ and arranged in increasing order. If $\frac{\Sigma_{t=1}^{k} a_t}{\Sigma_{t=1}^{m} a_t} \leq \eta$, then we have $f(\Omega', k) \leq \eta$, i.e., the feature subset of Ω' is a strongly correlated feature subset.*

The proof of Proposition 1 is straightforward and omitted here. This proposition gives us an upper bound of the objective function value for a given submatrix of Ω. For any submatrix $\Omega' = V \times \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ of Ω, we can examine the diagonal entries of the covariance matrix $C_{\Omega'}$ of Ω' to get the upper bound of the objective function. The computational cost of calculating this upper bound is much less than that of evaluating the objective function value directly by PCA. Therefore, before evaluating the objective function value of a candidate feature subset, we can check the upper bound in Proposition 1. If the upper bound is no greater than the threshold η, then we know that the candidate is a strongly correlated feature subset without performing PCA on its covariance matrix.

19.4.2 Choosing the Subsets of Points

In the previous section, we discussed the procedure of generating candidate feature subsets. A feature subset may be strongly correlated only on a subset of the data points. As discussed in Section 19.3.1, the monotonicity property does not hold for

the point subsets. Therefore, some heuristic must be used in order to avoid performing PCA on all possible subsets of points for each candidate feature subset. In this subsection, we discuss the heuristics that can be used for choosing the subset of points.

19.4.2.1 A Successive Point Deletion Heuristic

For a given candidate feature subset, if it is not strongly correlated on all data points, we can delete the points successively in the following way.

Suppose that $\Omega' = \{\mathbf{f}_{i_1}, \ldots, \mathbf{f}_{i_m}\} \times \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ is a submatrix of Ω and $f(\Omega', k) > \eta$, i.e., the features of Ω' is not strongly correlated on all data points. Let $\Omega'_{\setminus \mathbf{p}_a}$ be the submatrix of Ω' by deleting point $\mathbf{p}_a$ ($\mathbf{p}_a \in \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$) from Ω'. This heuristic deletes the point $\mathbf{p}_a$ from Ω' such that $f(\Omega'_{\setminus \mathbf{p}_a}, k)$ has the smallest value compared to deleting any other point. We keep deleting points until the number of points in the submatrix reaches the ratio $n/N = \delta$ or the feature subset of Ω' turns out to be strongly correlated on the current point subset.

This is a successive greedy point deletion heuristic. In each iteration, it deletes the point that leads to the most reduction in the objective function value. This heuristic is time consuming, since in order to delete one point from a submatrix containing n points, we need to calculate the objective function value n times in order to find the smallest value.

19.4.2.2 A Distance-Based Point Deletion Heuristic

In this section, we discuss the heuristic used by CARE. It avoids calculating objective function value n times for deleting a single point from a submatrix containing n points.

Suppose that $\Omega' = \{\mathbf{f}_{i_1}, \ldots, \mathbf{f}_{i_m}\} \times \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ is a submatrix of Ω and $f(\Omega', k) > \eta$, i.e., the features of Ω' are not strongly correlated on all data points. Let S_1 be the subspace spanned by the k eigenvectors with the smallest eigenvalues and S_2 be the subspace spanned by the remaining $m-k$ eigenvectors. For each point $\mathbf{p}_a$ ($\mathbf{p}_a \in \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$), we calculate two distances: d_{a_1} and d_{a_2}. d_{a_1} is the distance between $\mathbf{p}_a$ and the origin in sub-eigenspace S_1 and d_{a_2} is the distance between $\mathbf{p}_a$ and the origin in sub-eigenspace S_2. Let the distance ratio $r_{\mathbf{p}_a} = d_{a_1}/d_{a_2}$. We sort the points according to their distance ratios and delete $(1-\delta)N$ points that have the largest distance ratios.

The intuition behind this heuristic is that we try to reduce the variance in subspace S_1 as much as possible while retaining the variance in S_2.

Using the running data set shown in Fig. 19.5, for feature subset $\{\mathbf{f}_2, \mathbf{f}_7, \mathbf{f}_9\}$, the deleted points are shown as stars in Fig. 19.6a and b using the two different heuristics described above. The reestablished linear correlations are $2\mathbf{f}_2 + 5.9\mathbf{f}_7 + 3.8\mathbf{f}_9 = 0$ (successive), and $2\mathbf{f}_2 + 6.5\mathbf{f}_7 + 2.9\mathbf{f}_9 = 0$ (distance based). Note that the embedded linear correlation is $2\mathbf{f}_2 + 6\mathbf{f}_7 + 3\mathbf{f}_9 = 0$. As we can see from the figures, both methods choose almost the same point subsets and correctly reestablish the embedded linear correlation.

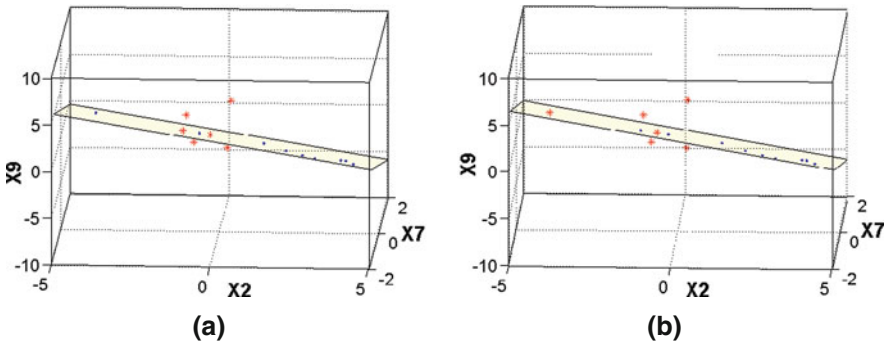


Fig. 19.6 Points deleted using different heuristics. (**a**) Successive point deletion; (**b**) Distance-based point deletion

The distance-based heuristic is more efficient than the successive approach since it does not have to evaluate the value of the objective function many times for each deleted point.

As a summary of Section 19.4, CARE adopts the depth-first search strategy to enumerate the candidate feature subsets. If a candidate feature subset is not strongly correlated on all data points, then CARE applies the distance-based point deletion heuristic to find the subset of points on which the candidate feature subset may have stronger correlation. If a candidate turns out to be a linear reducible subspace, then all its supersets can be pruned.

19.5 The REDUS Algorithm

In this section, we present REDUS algorithm which detects the (nonlinear) maximum reducible subspaces (Definition 6). We first give a short introduction to the intrinsic dimensionality estimator. Then we present the algorithms for finding the overall reducible subspace and the maximum reducible subspace, respectively.

19.5.1 Intrinsic Dimensionality Estimator

To find the overall reducible subspace in the data set, we adopt correlation dimension [33, 34], which can measure both linear and nonlinear intrinsic dimensionality, as our intrinsic dimensionality estimator since it is computationally more efficient than other estimators while its quality of estimation is similar to others. In practice, we observe that correlation dimension satisfies Properties 2 and 3, although we do not provide the proof here. In what follows, we give a brief introduction of correlation dimension.

Let Y be a feature subspace of the data set, i.e., $Y \subseteq \Omega_F$. Suppose that the number of points N in the data set approaches infinity. Let $dis(p_i, p_j, Y)$ represent the distance between two data points p_i and p_j in feature subspace Y. Let $B_Y(p_i, r)$

be the subset of points contained in a ball of radius r centered at point p_i in subspace Y, i.e.,

$$B_Y(p_i, r) = \{p_j | p_j \in \Omega_P, dis(p_i, p_j, Y) \leq r\}.$$

The average fraction of pairs of data points within distance r is

$$C_Y(r) = \lim_{N\to\infty} \frac{1}{N^2} \sum_{p_i \in \Omega_P} |B_Y(p_i, r)|.$$

The *correlation dimension* of Y is then defined as

$$\mathrm{ID}(Y) = \lim_{r,r'\to 0} \frac{\log[C_Y(r)/C_Y(r')]}{\log[r/r']}.$$

In practice, N is a finite number. C_Y is estimated using $\frac{1}{N^2} \sum_{p_i \in Y_P} |B(p_i, r)|$. The correlation dimension is the growth rate of the function $C_Y(r)$ in log–log scale, since $\frac{\log[C_Y(r)/C_Y(r')]}{\log[r/r']} = \frac{\log[C_Y(r)] - \log[C_Y(r')]}{\log r - \log r'}$. The correlation dimension is estimated using the slope of the line that best fits the function in least squares sense.

The intuition behind the correlation dimension is following. For points that are arranged on a line, one expects to find twice as many points when doubling the radius. For the points scattered on two-dimensional plane, when doubling the radius, we expect the number of points to increase quadratically. Generalizing this idea to m-dimensional space, we have $C_Y(r)/C_Y(r') = (r/r')^m$. Therefore, the intrinsic dimensionality of feature subspace Y can be simply treated as the growth rate of the function $C_Y(r)$ in log–log scale.

19.5.2 Finding Overall Reducible Subspace

The following theorem sets the foundation for the efficient algorithm to find the overall reducible subspace.

Theorem 2 *Suppose that $Y \subseteq \Omega_F$ is a maximum reducible subspace and $V \subset Y$ is its core space. We have $\forall U \subset Y$ ($|U| = |V|$), U is also a core space of Y.*

Proof We need to show that U satisfies the criteria in Definition 6. Let $V = \{f_{v_1}, f_{v_2}, \ldots, f_{v_m}\}$ and $U = \{f_{u_1}, f_{u_2}, \ldots, f_{u_m}\}$.

Since $U \subset Y$, from the definition of reducible subspace, U is non-redundant, and for every $f_{u_i} \in U$, $\Delta\mathrm{ID}(V, f_{u_i}) \leq \epsilon$. For every $f_a \in Y$, we have $\Delta\mathrm{ID}(V, f_a) \leq \epsilon$. Thus from Property 3, we have $\Delta\mathrm{ID}(U, f_a) \leq \epsilon$. Similarly, for every $f_b \notin Y$, $\Delta\mathrm{ID}(V, f_b) > \epsilon$. Thus $\Delta\mathrm{ID}(U, f_a) > \epsilon$.

Therefore, U is also a core space of Y.

Theorem 2 tells us that any subset $U \subset Y$ of size $|V|$ is also a core space of Y.

Suppose that $\{Y_1, Y_2, \ldots, Y_S\}$ is the set of all maximum reducible subspaces in the data set and the overall reducible subspace is $OR = \bigcup_{i=1}^{S} Y_i$. To find OR, we can apply the following method. For every $f_a \in \Omega_F$, let $RF_{f_a} = \{f_b | f_b \in \Omega_F, b \neq a\}$ be the remaining features in the data set. We calculate $\Delta\text{ID}(RF_{f_a}, f_a)$. The overall reducible subspace $OR = \{f_a | \Delta\text{ID}(RF_{f_a}, f_a) \leq \epsilon\}$.We now prove the correctness of this method.

Corollary 2 $OR = \{f_a | \Delta\text{ID}(RF_{f_a}, f_a) \leq \epsilon\}$.

Proof Let f_y be an arbitrary feature in the overall reducible subspace. From Theorem 2, we have $\forall f_y \in Y_i \subseteq OR, \exists V_i \subset Y_i$ $(f_y \notin V_i)$, such that V_i is the core space of Y_i. Thus $\Delta\text{ID}(V_i, f_y) \leq \epsilon$. Since $f_y \notin V_i$, we have $V_i \subseteq RF_{f_y}$. From Property 2, we have $\Delta\text{ID}(RF_{f_y}, f_y) \leq \epsilon$.

Similarly, if $f_y \notin OR$, then $\Delta\text{ID}(RF_{f_y}, f_y) > \epsilon$.

Therefore, we have $OR = \{f_y | \Delta\text{ID}(RF_{f_y}, f_y) \leq \epsilon\}$.

The algorithm for finding the overall reducible subspace is shown in Algorithm 1 from Line 1 to Line 7. Note that the procedure of finding overall reducible subspace is linear to the number of features in the data set.

Algorithm 1 REDUS

Input: Dataset Ω, input parameters ϵ, n, and τ,
Output: Y: the set of all maximum reducible subspaces

```
1  OR = ∅;
2  for each f_a ∈ Ω_F do
3  |   RF_{f_a} = {f_b | f_b ∈ Ω_F, b ≠ a};
4  |   if ΔID(RF_{f_a}, f_a) ≤ ε then
5  |   |   OR = OR ∪ {f_a};
6  |   end
7  end
8  sample n points P = {p_{s_1}, p_{s_2}, ··· , p_{s_n}} from Ω.
9  for d = 1 to |OR| do
10 |   for each candidate core space C ⊂ OR (|C| = d) do
11 |   |   T = {f_a | f_a is strongly correlated with C, f_a ∈ OR, f_a ∉ C};
12 |   |   if T ≠ ∅ then
13 |   |   |   Y ← T;
14 |   |   |   update OR by removing from OR the features in T;
15 |   |   end
16 |   end
17 end
18 return Y.
```

19.5.3 Maximum Reducible Subspace

In this section, we present the second component of REDUS, i.e., identifying the maximum reducible subspaces from the overall reducible subspace found in the previous section.

19.5.3.1 Intrinsic Dimensionality-Based Method

From Definition 6 and Theorem 2, we have the following property concerning the reducible subspaces.

Corollary 3 *Let $Y_i \subseteq OR$ be a maximum reducible subspace, and $V_i \subset Y_i$ be any core space of Y_i. We have*

$$Y_i = \{f_a | \Delta\text{ID}(V_i, f_a) \leq \epsilon, f_a \in OR\}.$$

Therefore, to find the individual maximum reducible subspaces $Y_i \subseteq OR$ $(1 \leq i \leq S)$, we can use any core space $V_i \subset Y_i$ to find the other features in Y_i. More specifically, a *candidate* core space of size d is a feature subset $C \subset OR$ $(|C| = d)$. From size $d = 1$ to $|OR|$, for each candidate core space, let $T = \{f_a | \Delta\text{ID}(C, f_a) \leq \epsilon, f_a \in OR, f_a \notin C\}$. If $T \neq \emptyset$, then T is a maximum reducible subspace with core space of size d. The overall reducible subspace OR is then updated by removing the features in T. Note that the size of $|OR|$ decreases whenever some maximum reducible subspace is identified. We now prove the correctness of this method.

Corollary 4 *Any candidate core space is non-redundant.*

Proof It is easy to see any candidate core space of size 1 is non-redundant. Now, assume that all candidate core spaces of size $d - 1$ are non-redundant; we show all candidate core spaces of size d are non-redundant. We prove this by contradiction.

Let $V = \{f_{v_1}, f_{v_2}, \ldots, f_{v_d}\}$ be an arbitrary candidate core space of size d. Without loss of generality, assume that f_d is the redundant feature in V. Let $V' = \{f_1, f_2 \ldots, f_{v_{d-1}}\}$. We have $\Delta\text{ID}(V', f_{v_d}) \leq \epsilon$. Since $|V'| = d - 1$; V' is non-redundant according to the assumption. Moreover, we have $T = \{f_a | \Delta\text{ID}(V', f_a) \leq \epsilon, f_a \in OR, f_a \notin V'\} \neq \emptyset$, since $f_{v_d} \in T$. Therefore, $f_{v_d} \in T$ would have been removed from OR before the size of the candidate core spaces reaches d. This contradicts the assumption of f_{v_d} being in the candidate core space V. Therefore, we have that any candidate core space is non-redundant.

Corollary 5 *Let C be a candidate core space. If $\exists f_a \in OR$ such that $\Delta\text{ID}(C, f_a) \leq \epsilon$, then C is a true core space of some maximum reducible subspace in OR.*

Proof Let $Y = \{f_y | \Delta\text{ID}(C, f_y) \leq \epsilon, f_y \in OR\}$. Following the process of finding OR, we know that Y includes all and only the features in Ω_F that are strongly correlated with C. Thus $\exists C \subset Y$, such that C satisfies criterion (1) in Definition 5 and criterion (2) in Definition 6. Moreover, according to Corollary 4, C is non-redundant. Hence C also satisfies criterion (2) of Definition 5. Thus Y is a maximum reducible subspace with core space C.

In this method, for each candidate core space, we need to calculate $\Delta\text{ID}(C)$ and $\Delta\text{ID}(C \cup \{f_a\})$ for every $f_a \in OR$ in order to get the value of $\Delta\text{ID}(C, f_a)$. However, the intrinsic dimensionality calculation is computationally expensive. Since the intrinsic dimensionality estimation is inherently approximate, we propose in the

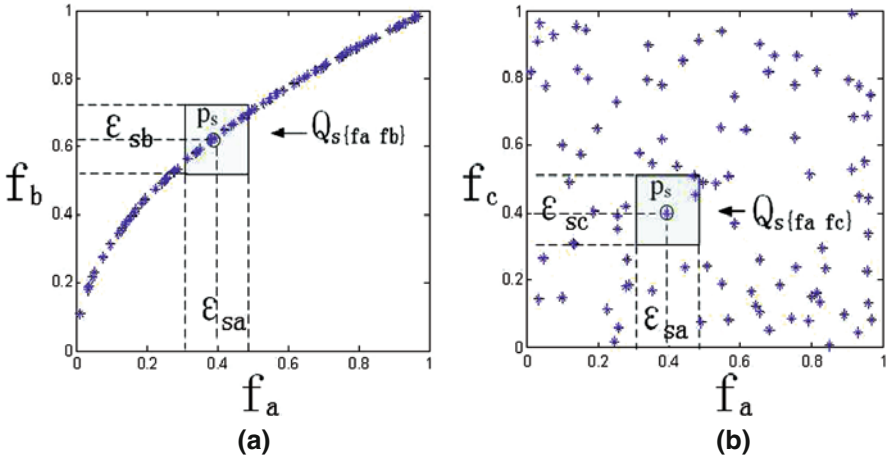


Fig. 19.7 Point distributions in correlated feature subspace and uncorrelated feature subspace. (**a**) strongly correlated features; (**b**) uncorrelated features

following section a method utilizing the point distribution in feature subspaces to distinguish whether a feature is strongly correlated with a core space.

19.5.3.2 Point Distribution-Based Method

After finding the overall reducible subspace OR, we can apply the following heuristic to examine if a feature is strongly correlated with a feature subspace. The intuition behind our heuristic is similar to the one behind the correlation dimension.

Assume that the number of data points N in the data set approaches infinity and the features in the data set are normalized so that the points are distributed from 0 to 1 in each dimension. Let $p_s \in \Omega_P$ be an arbitrary point in the data set, and $0 < l < 1$ be a natural number. Let ξ_{sy} represent the interval of length l on feature f_y centered at p_s. The expected number of points within the interval ξ_{sy} is lN. For d features $C = \{f_{c_1}, f_{c_2}, \ldots, f_{c_d}\}$, let Q_{sC} be the d-dimensional hypercube formed by the intervals ξ_{sc_i} ($f_{c_i} \in C$). If the d features in C are totally uncorrelated, then the expected number of points in Q_{sC} is $l^d N$. Let f_m be another feature in the data set, and $C' = \{f_{c_1}, f_{c_2}, \ldots, f_{c_d}, f_m\}$. If f_m is determined by $\{f_{c_1}, f_{c_2}, \ldots, f_{c_d}\}$, i.e., f_m is strongly correlated with C, then C' has intrinsic dimensionality d. The expected number of points in the d-dimensional hypercube, $Q_{sC'}$, which is embedded in the $(d+1)$-dimensional space of C', is still $l^d N$. If, on the other hand, f_m is uncorrelated with any feature subspace of $\{f_{c_1}, f_{c_2}, \ldots, f_{c_d}\}$, then C' has dimensionality $d+1$, and the expected number of points in the $(d+1)$-dimensional hypercube $Q_{sC'}$ is $l^{(d+1)}N$. The difference between the number of points in the cubes of these two cases is $l^d(1-l)N$.

Figure 19.7a,b show two examples on two-dimensional spaces. In both examples, $d = 1$ and $C = \{f_a\}$. In Fig. 19.7a, feature f_b is strongly correlated with f_a. Feature f_c is uncorrelated with f_a, as shown in Fig. 19.7b. The randomly sampled point

p_s is at the center of the cubes $Q_{s\{f_a,f_b\}}$ and $Q_{s\{f_a,f_c\}}$. The point density in cube $Q_{s\{f_a,f_b\}}$ is clearly much higher than the point density in cube $Q_{s\{f_a,f_c\}}$ due to the strong correlation between f_a and f_b.

Therefore, for each candidate core space, we can check if a feature is correlated with it in the following way. We randomly sample n points $P = \{p_{s_1}, p_{s_2}, \ldots, p_{s_n}\}$ from the data set. Suppose that $C = \{f_{c_1}, f_{c_2}, \ldots, f_{c_d}\}$ is the current candidate core space. For feature $f_a \in OR$ ($f_a \notin C$), let $C' = \{f_{c_1}, f_{c_2}, \ldots, f_{c_d}, f_a\}$. Let $\delta_{s_i C'}$ represent the number of points in the cube $Q_{s_i C'}$. $P' = \{p_{s_i} | \delta_{s_i C'} \geq l^{(d+1)} N\}$ is the subset of the sampled points such that the cube centered at them have more points than expected if f_a is uncorrelated with C. We say f_a is strongly correlated with C if $\frac{|P'|}{|P|} \geq \tau$, where τ is a threshold close to 1.

Concerning the choice of l, we can apply the following reasoning. If we let $l = \left(\frac{1}{N}\right)^{\frac{1}{d+1}}$, then the expected number of points in the cube $Q_{s_i C'}$ is 1, if f_a is uncorrelated with C. If f_a is correlated with C, then the expected number of points in the cube $Q_{s_i C'}$ is greater than 1. In this way, we can set l according to the size of the candidate core space.

The second step of REDUS is shown in Algorithm 1 from Line 8 to Line 18. Note that in the worst case, the algorithm needs to enumerate all possible feature subspaces. However, in practice, the algorithm is very efficient since once an individual reducible subspace is found, all its features are removed. Only the remaining features need to be further examined.

19.6 Experiments

In this section, we present the experimental results of CARE and REDUS on both synthetic and real data sets. Both algorithms are implemented using Matlab 7.0.4. The experiments are performed on a 2.4 GHz PC with 1G memory running WindowsXP system.

19.6.1 Synthetic Data

We evaluate CARE and REDUS on different synthetic data sets.

19.6.1.1 CARE

To evaluate the effectiveness of the CARE, we generate a synthetic data set of 100 features and 120 points in the following way. The data set is first populated with randomly generated points for each one of the 100 features. Then we embedded three local linear correlations into the data set as described in Table 19.4. For example, on points $\{\mathbf{p}_1, \ldots, \mathbf{p}_{60}\}$ we create local linear correlation $\mathbf{f}_{50} - \mathbf{f}_{20} + 0.5\mathbf{f}_{60} = 0$. Gaussian noise with mean 0 and variance 0.01 is added into the data set.

Table 19.4 Local linear correlations embedded in the data set

Point subsets	Local linear correlations
$\{\mathbf{p}_1, \cdots, \mathbf{p}_{60}\}$	$\mathbf{f}_{50} - \mathbf{f}_{20} + 0.5\mathbf{f}_{60} = 0$
$\{\mathbf{p}_{30}, \cdots, \mathbf{p}_{90}\}$	$\mathbf{f}_{40} - \mathbf{f}_{30} + 0.8\mathbf{f}_{80} - 0.5\mathbf{f}_{10} = 0$
$\{\mathbf{p}_{50}, \cdots, \mathbf{p}_{110}\}$	$\mathbf{f}_{15} - \mathbf{f}_{25} + 1.5\mathbf{f}_{45} - 0.3\mathbf{f}_{95} = 0$

We first show the comparison of CARE and full dimensional PCA. We perform PCA on the synthetic data set described above. To present the linear correlation discovered by PCA, we show the resulting hyperplanes determined by the three eigenvectors with the smallest eigenvalues. Each such hyperplane represents a linear correlation of all the features in the data set. Due to the large number of features, we only show the features with coefficients with absolute values greater than 0.2. The linear correlations reestablished by full dimensional PCA are shown in Table 19.5. Clearly, these are not the local linear correlations embedded in the data set.

Table 19.5 Linear correlations identified by full dimensional PCA

Eigenvectors	Linear correlations reestablished
$\mathbf{v}_1$ ($\lambda_1 = 0.0077$)	$0.23\mathbf{f}_{22} - 0.25\mathbf{f}_{32} - 0.26\mathbf{f}_{59} \approx 0$
$\mathbf{v}_2$ ($\lambda_2 = 0.0116$)	$0.21\mathbf{f}_{34} - 0.26\mathbf{f}_{52} \approx 0$
$\mathbf{v}_3$ ($\lambda_3 = 0.0174$)	$-0.22\mathbf{f}_6 - 0.29\mathbf{f}_8 + 0.22\mathbf{f}_{39}$
	$-0.23\mathbf{f}_{72} + 0.26\mathbf{f}_{93} \approx 0$

Table 19.6 shows the local linear correlations reestablished by CARE, with $k = 1$, $\eta = 0.006$, $\delta = 50\%$, and $max_s = 4$. As can be seen from the table, CARE correctly identifies the correlations embedded in the data set.

Table 19.6 Local linear correlations identified by CARE

$\mathbf{f}_{50} - 0.99\mathbf{f}_{20} + 0.42\mathbf{f}_{60} = 0$
$\mathbf{f}_{40} - 0.97\mathbf{f}_{30} + 0.83\mathbf{f}_{80} - 0.47\mathbf{f}_{10} = 0$
$\mathbf{f}_{15} - 0.9\mathbf{f}_{25} + 1.49\mathbf{f}_{45} - 0.33\mathbf{f}_{95} = 0$

Figure 19.8 shows the hyperplane representation of the local linear correlation, $\mathbf{f}_{40} - 0.97\mathbf{f}_{30} + 0.83\mathbf{f}_{80} - 0.47\mathbf{f}_{10} = 0$, reestablished by CARE. Since this is a three-dimensional hyperplane in four-dimensional space, we visualize it as a two-dimensional hyperplane in three-dimensional space by creating a new feature $(-0.83\mathbf{f}_{80} + 0.47\mathbf{f}_{10})$. As we can see from the figure, the data points are not clustered on the hyperplane even though the feature subsets are strongly correlated. The existing projected clustering algorithms [14, 15] try to find the points that are close to each other in the projected space. Therefore, they cannot find the strongly correlated feature subset as shown in this figure.

To evaluate the efficiency of CARE, we generate synthetic data sets as follows. Each synthetic data set has up to 500K points and 60 features, in which 40 linear correlations are embedded. Gaussian noise with mean 0 and variance 0.01 is added into the data set. The default data set for efficiency evaluation contains 5000 points

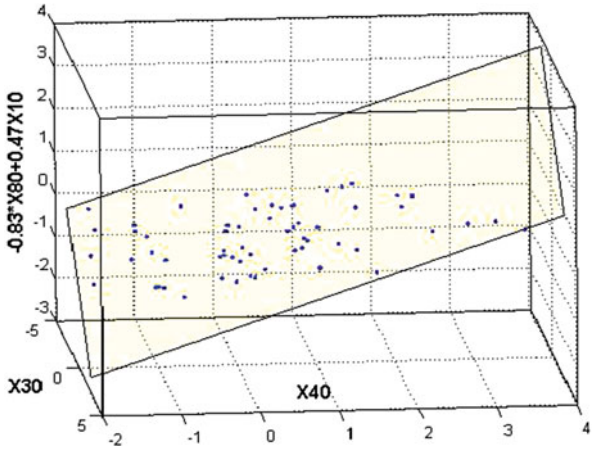


Fig. 19.8 The hyperplane representation of a local linear correlation reestablished by CARE

and 60 features if not specified otherwise. The default values for the parameters are $k = 1$, $\eta = 0.006$, $\delta = 50\%$, and $max_s = 4$.

Figure 19.9a–f show the efficiency evaluation results. Figure 19.9a shows that the running time of CARE is roughly quadratic to the number of features in the data set. Note that the theoretical worst case should be exponential when the algorithm has to check every subset of the features and data points. Figure 19.9b shows the scalability of CARE with respect to the number of points when the data set contains 30 features. The running time of CARE is linear to the number of data points in the data set as shown in the figure. This is due to the distance-based point deletion heuristic. As we can see from the figure, CARE finishes within reasonable amount of time for large data sets. However, since CARE scales roughly quadratically to the number of features, the actual runtime of CARE mostly depends on the number of features in the data set.

Figure 19.9c shows that the runtime of CARE increases steadily until η reaches certain threshold. This is because the higher the value of η, the weaker the correlations identified. After certain point, too many weak correlations meeting the criteria will be identified. Figure 19.9d demonstrates that CARE's runtime when varying δ. Figure 19.9e shows CARE's runtime with respect to different max_s when the data sets contain 20 features.

Figure 19.9f shows the number of patterns evaluated by CARE before and after applying the upper bound of the objective function value discussed in Section 19.4.

19.6.1.2 REDUS

As shown in Algorithm 1, REDUS generally requires three input parameters: ϵ, n, and τ. In the first step of finding the overall reducible subspace, ϵ is the threshold to

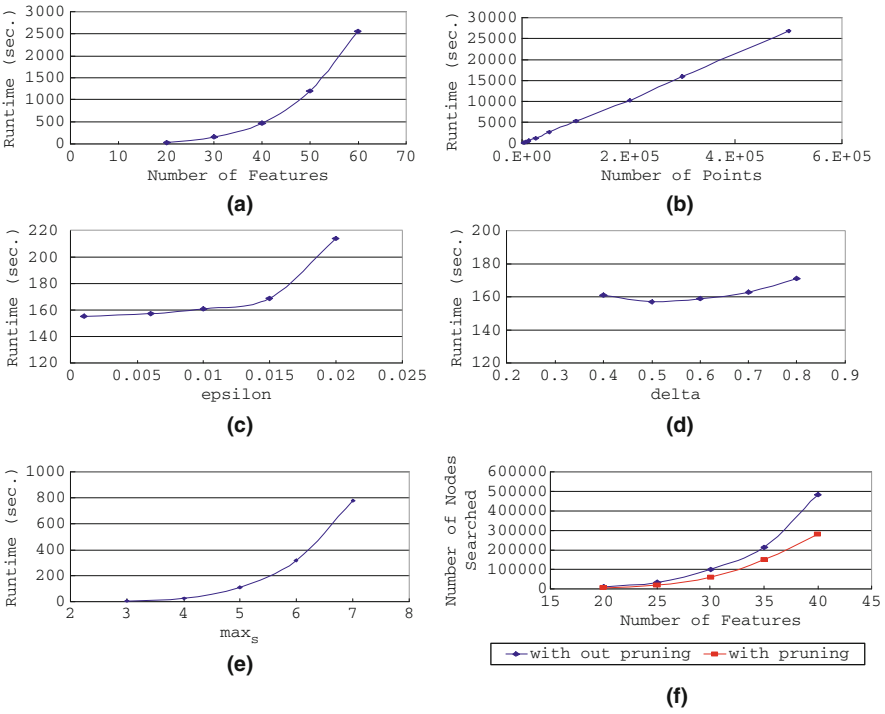


Fig. 19.9 CARE Efficiency evaluation. (**a**) Varying number of features; (**b**) Varying number of data points; (**c**) Varying η; (**d**) Varying δ; (**e**) Varying max_s; (**f**) Pruning effect of the upper bound of the objective function value

filter out the irrelevant features. Since features strongly correlated with some core space can only change intrinsic dimensionality a small amount, the value of ϵ should be close to 0. According to our experience, a good starting point is 0.1. After finding the reducible subspaces, the user can apply the standard dimensionality reduction methods to see if the are really correlated, and the adjust ϵ value accordingly to find stronger or weaker correlations in the subspaces. In all our experiments, we set ϵ between 0.002 and 0.25. In the second step, n is the point sampling size and τ is the threshold to determine if a feature is strongly correlated with a candidate core space. In our experiments, n is set to be 10% of the total number of data points in the data set and τ is set to be 90%.

We generate two synthetic data sets.

REDUS Synthetic Data Set 1

The first synthetic data set is as shown in Fig. 19.3. There are 12 features $\{f_1, f_2, \ldots, f_{12}\}$ and 1000 data points in the data set. Three reducible subspaces, a two-dimensional Swiss roll, a one-dimensional helix-shaped line, and

a two-dimensional plane, are embedded in different three-dimensional spaces, respectively. The overall reducible subspace is $\{f_1, f_2, \ldots, f_9\}$. Let c_i $(1 \leq i \leq 4)$ represent constants and r_j $(1 \leq j \leq 3)$ represent random vectors. The generating function of the Swiss roll is $t = \frac{3}{2}\pi(1 + 2r_1)$, $s = 21r_2$, $f_1 = t\cos(t)$, $f_2 = s$, and $f_3 = t\sin(t)$. The roll is then rotated 45° counterclockwise on feature space $\{f_2, f_3\}$. The helix-shaped line is generated by $f_4 = c_1 r_3$, $f_5 = c_2 \sin(r_3)$, and $f_6 = c_2 \cos(r_3)$. The two-dimensional plane is generated by $f_9 = c_3 f_7 + c_4 f_8$. The remaining three features $\{f_{10}, f_{11}, f_{12}\}$ are random vectors consisting of noise data points.

In the first step, with $\epsilon = 0.25$, REDUS successfully uncovers the overall reducible space. The parameter setting for the second step is $\tau = 90\%$, and point sampling size 10%. We run REDUS 10 times. In all 10 runs, REDUS successfully identifies the individual maximum reducible subspaces from the overall reducible subspace.

REDUS Synthetic Data Set 2

We generate another larger synthetic data set as follows. There are 50 features $\{f_1, f_2, \ldots, f_{50}\}$ and 1000 data points in the data set. There are three reducible subspaces: $Y_1 = \{f_1, f_2, \ldots, f_{10}\}$ reducible to a two-dimensional space, $Y_2 = \{f_{11}, f_{12}, \ldots, f_{20}\}$ reducible to a one-dimensional space, and $Y_3 = \{f_{21}, f_{22}, \ldots, f_{30}\}$ reducible to a two-dimensional space. The remaining features contain random noises. Figures 19.10a,b show two examples of the embedded correlations in three-dimensional subspaces. Figure 19.10a plots the point distribution on feature subspace $\{f_1, f_2, f_9\}$ of Y_1, and Fig. 19.10b plots the point distribution on feature subspace $\{f_{11}, f_{12}, f_{13}\}$ of Y_2.

We apply REDUS on this synthetic data set using various parameter settings. Table 19.7 shows the accuracy of finding the overall reducible subspace when ϵ takes different values. The recall is defined as $TP/(TP + FN)$, and the precision

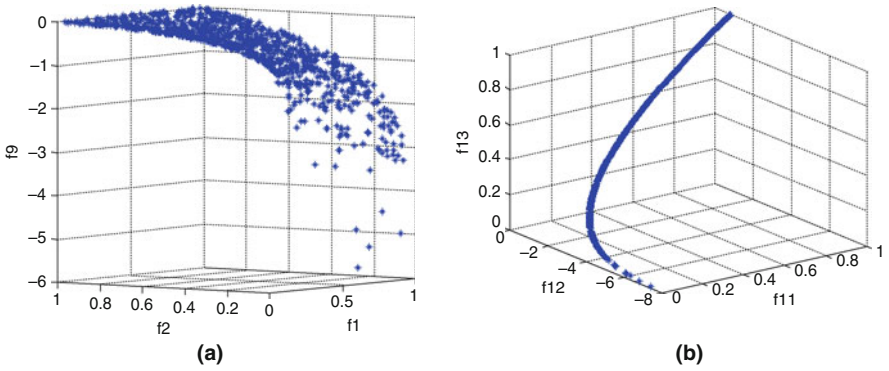


Fig. 19.10 Examples of embedded correlations in synthetic data set 2. (**a**) a correlation in Y_1 (**b**) a correlation in Y_2

Table 19.7 Accuracy of finding the overall reducible subspace when varying ϵ

ϵ	Precision	Recall
0.06	83%	100%
0.05	91%	100%
0.04	96%	100%
0.03	100%	100%
0.02	100%	100%
0.01	100%	100%
0	100%	90%

is defined as $TP/(TP + FP)$, where TP represents the number of true positive, FP represents the number of false positive, and FN represents the number of false negative. As we can see, REDUS is very accurate and robust to ϵ.

To evaluate the efficiency and scalability of REDUS, we apply it to *synthetic data set 2*. The default data set for efficiency evaluation contains 1000 points and 50 features if not specified otherwise. The default values for the parameters are the same as before.

Figure 19.11a shows the runtime of finding the overall reducible subspace when varying the number of data points. The runtime scales roughly quadratically. This is because when computing the correlation dimensions, we need to calculate all pairwise distances between the data points, which is clearly quadratic to the number of points.

Figure 19.11b shows that the runtime of finding the overall reducible subspace is linear to the number of features. This is because REDUS only scans every feature once to examine if it is strongly correlated with the subspace of the remaining features. This linear scalability is desirable for the data sets containing a large number of features.

Figure 19.12a,b show the runtime comparisons between using the correlation dimension as intrinsic dimensionality estimator and the point distribution heuristic to identify the individual maximum reducible subspaces from the overall reducible subspaces. Since the calculation of intrinsic dimensionality is relatively expensive, the program often cannot finish in a reasonable amount of time. Using the point

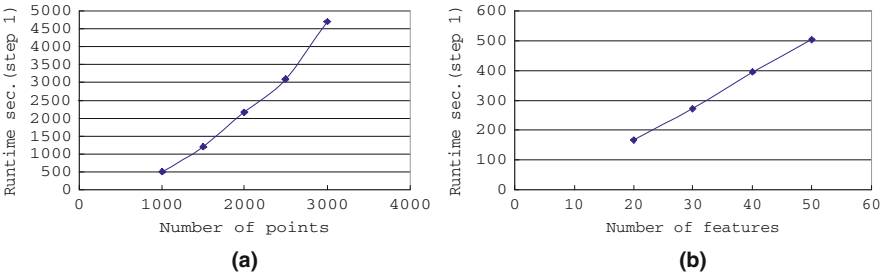


Fig. 19.11 REDUS efficiency evaluation of finding the overall reducible subspace. (**a**) Varying number of points; (**b**) Varying number of features

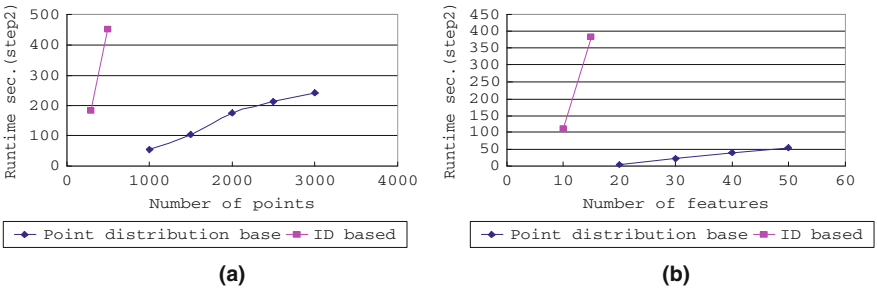


Fig. 19.12 REDUS Efficiency evaluation of identifying maximum reducible subspaces from the overall reducible subspace. (**a**) Varying number of points; (**b**) Varying number of features

distribution heuristics, on the other hand, is much more efficient and scales linearly to the number of points and features in the data set.

19.6.2 Real Data

We apply CARE on the mouse gene expression data provided by the School of Public Health at UNC. The data set contains the expression values of 220 genes in 42 mouse strains. CARE find eight strongly correlated gene subsets with parameter setting: $k = 1$, $\eta = 0.002$, $\delta = 50\%$, and $max_s = 4$. Due to the space limit, we show four of these eight gene subsets in Table 19.8 with their symbols and the corresponding GO annotations. As shown in the table, genes in each gene subset have consistent annotations. We also plot the hyperplanes of these strongly correlated gene subsets in three-dimensional space in Fig. 19.13a–d. As we can see from the figures, the data points are sparsely distributed in the hyperplanes, which again

Table 19.8 Strongly correlated gene subsets

Subsets	Gene IDs	GO annotations
1	Nrg4	Cell part
	Myh7	Cell part; intracelluar part
	Hist1h2bk	Cell part; intracelluar part
	Arntl	Cell part; intracelluar part
2	Nrg4	Integral to membrane
	Olfr281	Integral to membrane
	Slco1a1	Integral to membrane
	P196867	N/A
3	Oazin	Catalytic activity
	Ctse	Catalytic activity
	Mgst3	Catalytic activity
4	Hspb2	Cellular physiological process
	2810453L12Rik	Cellular physiological process
	1010001D01Rik	Cellular physiological process
	P213651	N/A

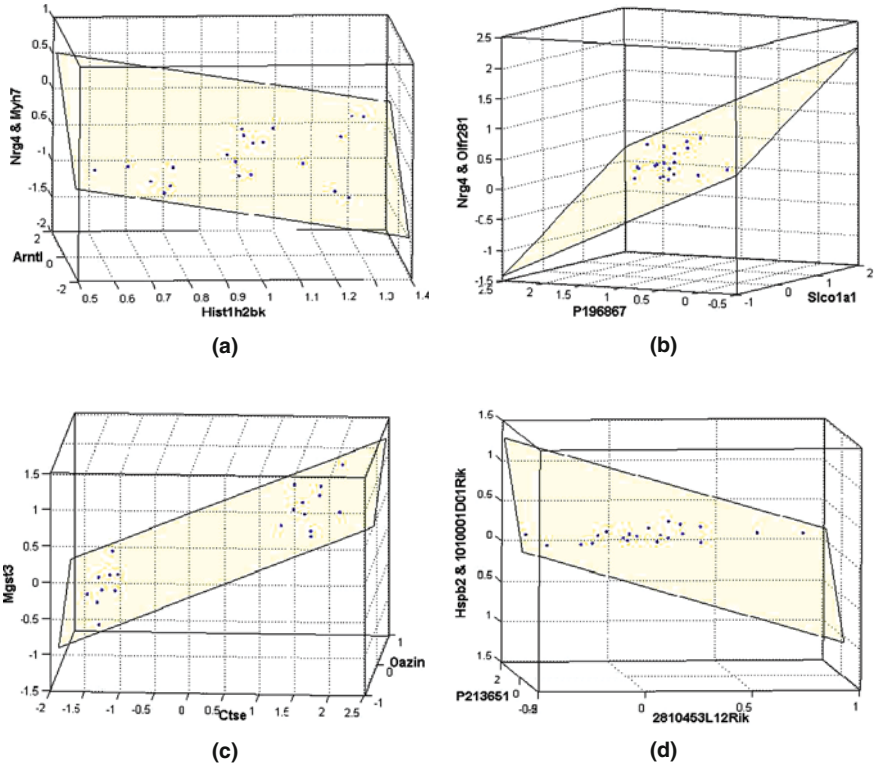


Fig. 19.13 Hyperplane representations of strongly correlated gene subsets. (**a**) Gene subset 1; (**b**) Gene subset 2; (**c**) Gene subset 3; (**d**) Gene subset 4

demonstrates CARE can find the groups of highly similar genes which cannot be identified by the existing projected clustering algorithms.

19.7 Conclusion

In this chapter, we investigate the problem of finding strongly correlated feature subspaces in high-dimensional data sets. The correlation can be linear or nonlinear. Such correlations hidden in feature subspace may be invisible to the global feature transformation methods. Utilizing the concepts of PCA and intrinsic dimensionality, we formalize this problem as the discovery of maximum reducible subspaces in the data set. Two effective algorithms, CARE and REDUS, are presented to find the reducible subspaces in linear and nonlinear cases, respectively. The experimental results show that both algorithms can effectively and efficiently find these interesting local correlations. These methods are powerful tools for identifying potential transcriptional modules and thus play an important role in many modeling biological networks.

References

1. M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns, *Proceedings of National Acadamy of Science USA*, 95:14863–14868, 1998.
2. V. Iyer and et. al. The transcriptional program in the response of human fibroblasts to serum. *Science*, 283:83–87, 1999.
3. L. Parsons, E. Haque, and H. Liu. Subspae clustering for high dimensional data: a review, In *KDD Explorations*, 6(1): 90–105, 2004.
4. A. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, 97: 245–271, 1997.
5. H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer, Boston, MA, 1998.
6. L. Yu and H. Liu, Feature selection for high-dimensional data: a fast correlation-based filter solution. In *Proceedings of International Conference on Machine Learning*, 856–863, 2003.
7. Z. Zhao and H. Liu. Searching for interacting features, In *The 20th International Joint Conference on AI*, 1156–1161, 2007.
8. M. Belkin and P. Niyogi. Şlaplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 2003.
9. T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer, 1996.
10. I. Borg and P. Groenen. *Modern multidimensional scaling*. Springer, New York, 1997.
11. I. Jolliffe. *Principal Component Analysis*. Springer, New York, 1986.
12. S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290 (5500):2323–2326, 2000.
13. J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290 (5500):2319–2323, 2000.
14. C. Aggarwal and P. Yu. Finding generalized projected clusters in high dimensional spaces. In *SIGMOD*, 2000.
15. E. Achtert, C. Bohm, H.-P. Kriegel, P. Kroger, and A. Zimek. Deriving quantitative models for correlation clusters. In *KDD*, 2006.
16. H. Wang, W. Wang, J. Yang, and Y. Yu. Clustering by pattern similarity in large data sets. In *SIGMOD*, 2002.
17. M. Ashburner et al. Gene ontology: tool for the unification of biology, *The gene ontology consortium, Nature Genetics*, 25:25–29, 2000.
18. X. Zhang, F. Pan, and W. Wang. Care: Finding local linear correlations in high dimensional data. In *ICDE*, 130–139, 2008.
19. K. Fukunaga. Intrinsic dimensionality extraction. *Classification, Pattern recongnition and Reduction of Dimensionality, Volume 2 of Handbook of Statistics*, pages 347–360, P. R. Krishnaiah and L. N. Kanal editors, Amsterdam, North Holland, 1982.
20. F. Camastra and A. Vinciarelli. Estimating intrinsic dimension of data with a fractal-based approach. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(10):1404–1407, 2002.
21. K. Fukunaga and D. R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 20(2):165–171, 1976.
22. E. Levina and P. J. Bickel. Maximum likelihood estimation of intrinsic dimension. *Advances in Neural Information Processing Systems*, 2005.
23. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD*, 94–105, 1998.
24. C. Aggarwal, J. Wolf, P. Yu, C. Procopiuc, and J. Park. Fast algorithms for projected clustering. In *SIGMOD*, 61–72, 1999.
25. C. Chen, A. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *SIGKDD*, 84–93, 1999.
26. D. Barbara and P. Chen. Using the fractal dimension to cluster datasets. In *KDD*, 260–264, 2000.

27. A. Gionis, A. Hinneburg, S. Papadimitriou, and P. Tsaparas. Dimension induced clustering. In *KDD*, 2005.
28. S. Papadimitriou, H. Kitawaga, P. B. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE*, 2003.
29. B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *ICDE*, 589, 2000.
30. A. Belussi and C. Faloutsos. Self-spacial join selectivity estimation using fractal concepts. *ACM Transactions on Information Systems*, 16(2):161–201, 1998.
31. C. Faloutsos and I. Kamel. Beyond uniformity and independence: analysis of r-trees using the concept of fractal dimension. In *PODS*, 1994.
32. G. Golub and A. Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, MD, 1996.
33. S. N. Rasband. *Chaotic Dynamics of Nonlinear Systems*. Wiley, 1990.
34. M. Schroeder. *Fractals, Chaos, Power Lawers: Minutes from an Infinite Paradise*. W. H. Freeman, New York, 1991.
35. R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge UK, 1985.
36. D. C. Lay. *Linear Algebra and Its Applications*. Addison Wesley, 2005.

Chapter 20
Functional Influence-Based Approach to Identify Overlapping Modules in Biological Networks

Young-Rae Cho and Aidong Zhang

Abstract The inherent, dynamic, and structural behaviors of complex biological networks in a topological perspective have been widely studied recently. These studies have attempted to discover hidden functional knowledge on a system level since biological networks provide insights into the underlying mechanisms of biological processes and molecular functions within a cell. Functional modules can be identified from biological networks as a sub-network whose components are highly associated with each other through links. Conventional graph-theoretic algorithms had a limitation in efficiency and accuracy on functional modules detection because of complex connectivity and overlapping modules. Whereas partition-based or hierarchical clustering methods produce pairwise disjoint clusters, density-based clustering methods that search densely connected sub-networks are able to generate overlapping clusters. However, they are not well applicable to identifying functional modules from typically sparse biological networks. Recently proposed functional influence-based approach effectively handles the complex but sparse biological networks, generating large-sized overlapping modules. This approach is based on the functional influence model, which quantifies the influence of a source vertex on each target vertex. The experiment with a real protein interaction network in yeast shows that this approach has better performance than other competing methods. A better understanding of higher-order organizations that are identified by functional influence patterns in biological networks can be explored in many practical biomedical applications.

20.1 Introduction

Biological networks, such as metabolic networks, protein interaction networks, and gene regulatory networks, are typically described as complex systems. Biological

A. Zhang (✉)
State University of New York, Buffalo, NY 14260, USA
e-mail: azhang@cse.buffalo.edu

Y.-R. Cho
Baylor University, Waco, TX 76798, USA
e-mail: young-rae_cho@baylor.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_20,

networks contain the information of biochemical reactions or biophysical interactions between molecular components at a certain environmental condition. Since molecular functions are performed by a sequence of such reactions and interactions, biological networks are valuable resources for characterizing functions of unknown genes or proteins and discovering functional pathways. Systematic analysis of biological networks has thus become a primary issue in bioinformatics research [42].

A functional module is defined as a maximal set of molecules that participate in the same function [22, 39]. It can be identified from biological networks as a sub-network whose components are highly associated with each other through links. In recent studies, it has been investigated that typical biological networks are modular in topology [4]. A wide range of graph-theoretic algorithms have been applied to biological networks for identifying functional modules [36]. However, they had a limitation in accuracy and efficiency because of the critical challenges as the following:

- Genome-wide biological networks are typically in a large scale and structured by complex connectivity. It results from numerous cross-links (i.e., interconnections) between potential functional modules. Conventional graph-theoretic algorithms are inappropriate to handle such large, complex networks because of their unscalability and inefficiency in runtime.
- A molecule may be involved in several different functional activities under different environmental conditions. Without separating each condition, a network includes overlapping modules, i.e., each vertex is a member of one or more cluster(s). Therefore, conventional partition-based clustering (e.g., k-means algorithm) or hierarchical clustering methods (e.g., agglomerative or divisive algorithms) are inaccurate in biological network analysis because they produce pairwise disjoint clusters.

Most graph clustering algorithms search densely connected sub-graphs. A typical example is the maximal clique algorithm for detecting fully connected sub-graphs. Because of the strict constraints of maximal cliques, relatively dense sub-graphs can be identified rather than complete sub-graphs by either using a density threshold or optimizing an objective density function [37]. Assorted algorithms using alternative density functions have been recently presented [3, 9, 30]. These methods have been frequently applied to biological networks because of the feasibility of generating overlapping clusters. However, they exclude sparsely connected vertices from the output clusters with high-density thresholds because the sparse connections decrease the density of clusters. In addition, they are likely to combine distinct clusters with low density thresholds because of numerous cross-links between clusters. Therefore, they are not able to detect functional modules with appropriate sizes from a biological network which is typically represented as a sparse but complex graph.

Recently, a novel approach based on the functional influence model has been proposed [11, 13]. This model is designed to quantify the functional influence of a source vertex on each target vertex in the input network. The edge weights and vertex connectivity on the path from the source to the target are significant factors for the quantification. The functional influence model is simulated by a flow-based

algorithm. The quantity of functional influence on each target vertex determines whether the target vertex is included in the same functional module with the source vertex. This approach has been tested with a real protein interaction network in yeast. The results demonstrate that this approach has better performance than other competing methods in terms of accuracy. Manually annotated functional categories from the MIPS database [27] are used as ground truth.

The remainder of this chapter is organized as follows. In Section 20.2, we survey previous graph clustering methods which have been applied to functional module detection from biological networks. In Section 20.3, the functional influence model and flow-based functional influence simulation algorithm are presented. In Section 20.4, the modularization algorithm including three steps is discussed. In Section 20.5, we show the experimental results in the real application to a protein interaction network.

20.2 Survey of Previous Module Detection Approaches

Previous graph clustering methods for detecting functional modules can be categorized into three groups: partition-based clustering, hierarchical clustering, and density-based clustering. First, the partition-based clustering approaches explore the best partition which separates the graph into several sub-graphs. Next, the hierarchical clustering approaches iteratively merge two sub-graphs that are the most similar (or closest) or recursively divide the graph or sub-graph by disconnecting the weakest links. Finally, the density-based approaches search densely connected sub-graphs. The previous methods in these three categories are summarized in the following subsections.

20.2.1 Partition-Based Clustering

20.2.1.1 Restricted Neighborhood Search Clustering (RNSC)

King et al. [26] proposed a cost-based local search algorithm modeled on the meta-heuristic search. The process begins with a list of random or user-specific clusters and calculates the cost of initial clusters by the cost function defined below. It then iteratively moves each node on the border of a cluster to an adjacent cluster in a random manner to find a lower cost. It finally detect the partition with the minimum cost.

The cost function is determined by the proportion of interconnections. Suppose we have a partition of a graph G. Let α_v be the number of interconnections that are linked from v to vertices in different clusters. The naive cost function is then defined as:

$$C_n(G) = \frac{1}{2} \sum_{v \in V} \alpha_v. \tag{20.1}$$

For a vertex v in G, let β_v be the total number of vertices in the cluster having v plus the number of vertices linked to v in different clusters. The scaled cost function is then defined as

$$C_n(G) = \frac{|V| - 1}{3} \sum_{v \in V} \frac{\alpha_v}{\beta_v}. \tag{20.2}$$

This measure reflects the size of the area that v influences in the cluster. Both cost functions seek to define a clustering scenario in which the vertices in a cluster are all connected to one another and there are no other connections between two clusters. Since the RNSC algorithm starts with a random partition, different runs on the same input data will generate different clustering results. As another weakness, this method requires the prior knowledge of the exact number of clusters existing in a network.

This method has been applied to finding protein complexes from protein interaction networks [26]. However, they needed to filter the output modules to find true protein complexes according to cluster size, cluster density, and functional homogeneity. Only the clusters that satisfy these three criteria have been considered as predicted functional modules.

20.2.1.2 Markov Clustering (MCL)

The Markov clustering algorithm (MCL) was designed specifically for the applications to simple and weighted graphs [15, 17]. The MCL algorithm finds clusters in a graph by a mathematical bootstrapping procedure. As an input graph, it uses a Markov matrix having an edge weight on each corresponding entry. This algorithm simulates random walks within the graph by alternating two operators: expansion and inflation. It starts with computing the random walks of the input graph, yielding a stochastic matrix. It then uses iterative rounds of the expansion operator, which takes the square of the stochastic matrix, and the inflation operator, which raises each matrix entry to a given power and then re-scales the matrix to return it to a stochastic state. This process continues until there is no further change in the matrix.

Expansion and inflation are used iteratively in the MCL algorithm to enhance the graph where it is strong and to diminish it where it is weak, until equilibrium is reached. Importantly, this algorithm has its "bootstrapping" nature, retrieving cluster structure via the imprint made by this structure on the flow process. Since the algorithm is fast and very scalable, it can be well applicable to real biological networks. Additionally, its accuracy is not compromised by the edges between different clusters. A recent study [7] has demonstrated that the MCL method is superior for extracting protein complexes from protein interaction networks even though it is not able to generate overlapping clusters.

20.2.2 Hierarchical Clustering

20.2.2.1 Bottom-Up Approaches

The bottom-up hierarchical clustering approaches start from single-vertex clusters and iteratively merge the closest vertices or clusters into a super-cluster. For the iterative merging, the similarity or distance between two vertices or two clusters should be measured, for example, the similarity from the reciprocal of the shortest path distance between two vertices [33] and the similarity from the statistical significance of common interacting partners [20, 35]. Other advanced distance metrics such as the Czekanowski-Dice distance [8] have also been used for this task. The Czekanowski-Dice distance D between two vertices i and j is described as

$$D(i, j) = \frac{|\mathrm{Int}(i) \Delta \mathrm{Int}(j)|}{|\mathrm{Int}(i) \cup \mathrm{Int}(j)| + |\mathrm{Int}(i) \cap \mathrm{Int}(j)|}, \tag{20.3}$$

where $\mathrm{Int}(i)$ denotes the set of neighboring vertices directly connected to i including the vertex i itself, and Δ represents the operator for symmetric difference between two sets.

Two sub-graphs to be merged can be selected by an optimization process. The similarity or distance can be measured in two steps to improve the accuracy. The UVCLUSTER algorithm [2] first uses the shortest path distances as primary distances to apply the agglomerative hierarchical clustering. Next, based on the clustering results, it calculates the secondary distances. As a greedy optimization algorithm [29], two sub-graphs to be merged can be found on each iteration by searching the best modularity. In this work, the modularity Q of a graph is defined as

$$Q = \sum_i \left(e_{ii} - (\Sigma_j e_{ij})^2\right), \tag{20.4}$$

where e_{ii} is the number of intra-connecting edges within a cluster i, and e_{ij} is the number of interconnecting edges between two clusters i and j. The super paramagnetic clustering (SPC) method [38] is another example of iterative merging. The most similar pair of vertices can be selected from identical ferromagnetic spins.

These approaches are appealing for real biological applications because biological functions are also described by hierarchical ordering in general. Despite the advantage of building the potential hierarchy of modular components, these approaches may not have a meaningful guidance of the halting point in the merging process to yield true functional modules.

20.2.2.2 Top-Down Approaches

The top-down hierarchical clustering approaches have the opposite procedure, starting from one cluster including all vertices in a graph and recursively dividing it. The recursive minimum-cut algorithm [21] is a typical example in this category. However, it is computationally expensive to find the minimum number of cut in a

complex system. Thus, the vertices or edges to be removed for the graph division can be selected in alternative ways. For example, they can be iteratively found using the betweenness measure [28] which gives a high score to the edge located between potential modules. The betweenness C_B of an edge e is calculated by the fraction of the shortest paths passing through the edge, i.e.,

$$C_B(e) = \sum_{s \neq t \in V, e \in E} \frac{|\rho_{st}(e)|}{|\rho_{st}|}, \tag{20.5}$$

where $\rho_{st}(e)$ is the set of the shortest paths between two vertices s and t, which are passing through the edge e, and ρ_{st} is the set of all shortest paths between s and t. Iterative elimination of the edges with the highest betweenness divides a graph into two or more sub-graphs, and the iteration is recursively proceeded into each sub-graph to detect final clusters [18]. While the betweenness assesses the global connectivity pattern for each vertex or edge, the local connectivity can be considered to select the interconnecting edges to be cut. For instance, the edge is selected by the smallest rate of common neighbors between two ending vertices [31].

These approaches can reveal the global view of the hierarchical structure. Previously, the betweenness-based hierarchical method has been popularly used in biological network analysis [16, 23]. However, finding the correct dividing points is the most crucial and time-consuming process for the application to large-sized, complex biological networks. In addition, as a critical shortcoming, these top-down hierarchical clustering approaches are sensitive to noisy data, which are frequently occurred in real biological networks.

20.2.3 Density-Based Clustering

20.2.3.1 Molecular Complex Detection (MCODE)

Molecular complex detection (MCODE) [3] is an effective approach for detecting densely connected regions in large biological networks. This method weights a vertex by local neighborhood density, chooses a few seeds with a high weight, and isolates the dense regions according to given parameters. The MCODE algorithm operates in three steps: vertex weighting, protein complex (cluster) generation, and optional post-processing step to filter or add vertices to the resulting clusters according to certain connectivity criteria.

In the first step, all vertices are weighted based on their local density using the highest k-core of the vertex neighborhood. The k-core of a graph is defined as the maximum sub-graph if every vertex has at least k links [41]. It is obtained by pruning all the vertices with a degree less than k. Thus, if a vertex v has degree d_v and it has n neighbors with degree less than k, then the degree of v becomes $d_v - n$. It will also be pruned if $k > (d_v - n)$. The core-clustering coefficient of a vertex v is defined as the density of the highest k-core of the vertices connected directly to v, together with v itself. For each vertex v, its weight w_v is

$$w_v = k \times d, \tag{20.6}$$

where d is the density of the highest k-core graph from the set of vertices including all the vertices directly connected with v and the vertex v itself. The clustering coefficient [40] is a traditional metric to quantify how well a vertex affects the local denseness. The clustering coefficient $c(v_i)$ of a vertex v_i is formulated as the ratio of the number of actual edges between direct neighbors of v_i to the number of all possible edges between them:

$$c(v_i) = \frac{|\{(v_j, v_k)|v_j \in N(v_i), v_k \in N(v_i), j \neq k\}|}{|N(v_i)| \times (|N(v_i) - 1|)}. \tag{20.7}$$

Comparing to this conventional clustering coefficient, the core-clustering coefficient in the MCODE algorithm amplifies weighting of heavily connected regions while removing many sparsely connected vertices.

The second step of the algorithm is the cluster generation. With a vertex-weighted graph as an input, a sub-graph with the highest-weighted vertex is selected as a seed. Once a vertex is included, its neighbors are recursively inspected to determine if they are a part of the cluster. The seed is then expanded to a cluster until it reaches a density threshold of the cluster. By checking a vertex more than once, overlapping clusters can be yielded. This process stops when no additional vertices can be added to the cluster. The vertices included in the cluster are marked as examined. This process is repeated for the next highest unexamined weighted vertex in the graph.

This method has been specifically devised to detect protein complexes in protein interaction networks [3]. In the experiment with the yeast protein interaction network, MCODE effectively located the densely connected regions as protein complexes based solely on the connectivity.

20.2.3.2 Clique Percolation

Derenyi et al. [14] introduced a novel process of *k-clique percolation*, along with the associated concepts of *k-clique adjacency* and *k-clique chain*. Two k-cliques are adjacent if they share $(k - 1)$ vertices, where k is the number of vertices in each clique. A k-clique chain is a sub-graph comprising the union of a sequence of adjacent k-cliques. A *k-clique percolation cluster* is thus a maximal k-clique chain. The k-clique percolation cluster is equivalent to a regular percolation cluster in the k-clique adjacency graph, where the vertices represent the k-cliques of the original graph, and there is an edge between two vertices if the corresponding k-cliques are adjacent. Using a heuristic approach, Derenyi et al. have found that the percolation transition of k-cliques in random graphs takes place when the probability of two vertices being connected by an edge reaches the threshold $p_c(k)$, where

$$p_c(k) = \frac{1}{\big((k-1) \cdot |V|\big)^{1/(k-1)}}. \tag{20.8}$$

The key advantage of the clique percolation method is its ability to identify overlapping clusters. Palla et al. [30] have tested the clique percolation method in the yeast protein interaction network. Through this experiment, they determined that the cumulative distribution of module size follows a power law with an exponent of -1, approximately. They additionally observed that the cumulative distribution of overlap size, which is the number of vertices shared in two modules, is close to a power law with a somewhat larger exponent.

20.2.3.3 Complex Overlap Decomposition (COD)

Zotenko et al. [43] proposed a graph theoretical method, called Complex Overlap Decomposition (COD), to detect overlapping clusters. This method is based on the concepts of chordal graphs and cograph. A chord in a graph is any edge that connects two non-consecutive vertices of a cycle. A chordal graph is a graph which does not contain chordless cycles of length greater than three. Every chordal graph has a corresponding clique tree representation. The topology of the clique tree is determined by the structure of overlaps between maximal cliques in a graph. The clique tree thus captures the structure of overlaps. Since some biological networks are not chordal, the COD method uses the edge addition strategy. A pair of vertices are called weak siblings if and only if they share exactly the same set of neighbors, but are not directly connected to each other. The COD method takes the first step toward delineating functional groups by connecting every pair of weak siblings. The second step is to obtain all clique trees from the modified graph if it is chordal. Otherwise, the process stops. However, since maximal cliques may not always be the best way to represent functional groups, a new concept, cographs, is introduced. A cograph is characterized by the absence of an induced sub-graph which has a path of length-4. For each cograph, there is a Boolean expression which describes all maximal cliques in the graph. Finally, each clique tree is extended to a "Tree of Complexes" representation of the original graph by projecting each clique to a functional group in the original graph.

This approach has been devised for identifying functional groups in protein interaction networks and applied to TNFα/NFκB and pheromone signaling pathways [43]. The experimental results show that the COD method successfully identified some proteins in the same functional groups and some functional groups within more than one protein complex.

20.2.3.4 Seed Growth

Altaf-Ul-Amin et al. [1] recently developed another density-based clustering method. This algorithm needs an input of the associated matrix of a graph. It ranks the vertices by their degrees from top to bottom. The vertex with the highest degree will be selected as the first seed node. The cluster then starts from the seed node and grows gradually by adding vertices one by one from its neighbors. It is important to add neighbors to the cluster by priority to guide the cluster formation in a proper way. The priority is determined based on two measures: (1) the sum of the

edge weights between a neighbor and each of the vertices in the cluster and (2) the number of edges between a neighbor and each of the vertices in the cluster. Furthermore, it checks another two points. First, the density of the cluster should be below a threshold. Second, it should be investigated whether the vertex is a part of the cluster by evaluating the cluster property. The cluster property cp_{nk} of a vertex n with respect to any cluster k of density d_k and size $|N_k|$ is defined as

$$cp_{nk} = \frac{|E_{nk}|}{d_k \times |N_k|}, \tag{20.9}$$

where $|E_{nk}|$ is the total number of edges between the vertex n and each of the vertices in k. Once a cluster is generated, the graph is updated by removing the cluster. The next cluster can be generated from the remaining graph. However, to produce overlapping clusters, vertices should be added starting from their first neighbors in the original graph, not in the remaining graph, until there are no vertices left in the graph.

The performance of this method has been tested in real protein interaction networks [1]. As a result, a significant number of predicted complexes have been matched with known protein complexes. In addition, the quality of the predicted complexes has been evaluated by the ratio of the interactions that occur between protein pairs with similar functions. It has been observed that the percentage of the interactions between protein pairs with common functions is higher in high-density complexes.

20.3 Modeling and Simulation of Functional Influence

As a novel direction for effective analysis of biological networks, a functional influence model and flow-based functional influence simulation algorithm have been presented [11, 13]. The key concept of this model is that a molecule, such as a gene, protein, and enzyme, has a functional influence on another molecule through links in a biological network. This approach requires a weighted, undirected graph as the input network. The intensity or reliability of each link should be assessed and assigned into the corresponding edge as its weight. The details of the functional influence model and simulation algorithm will be discussed in the following subsections.

20.3.1 Functional Influence Model

The functional influence model is designed to describe the propagation of functional influence of a molecule over the entire network. This model is thus implemented by simulating the quantity of the influence of a vertex $v_i \in V$ on the others $v_j \in V$, $j \neq i$, in the input graph $G(V, E)$. The primary assumption is that each vertex contains the self-information which can be propagated through all links. As a central

component of this model, the path strength S of a path p is defined as the product of the weighted probabilities that each vertex on p chooses the succeeding vertex. The weighted probability from v_i to v_j is the ratio of the weight between v_i and v_j to the sum of the weights between v_i and its neighbors directly connected:

$$S(p) = \lambda \prod_{i=0}^{n-1} \frac{w_{i(i+1)}}{d^{wt}(v_i)}, \tag{20.10}$$

where $p = \langle v_0, v_1, \ldots, v_n \rangle$. v_0 is the start vertex and v_n is the end vertex of p. $w_{i(i+1)}$ denotes the weight of the edge between v_i and $v_{(i+1)}$, which is normalized into the range between 0 and 1. $d^{\text{wt}}(v_i)$ represents the shape parameter that indicates the weighted degree of the vertex v_i. The weighted degree of v_i is the sum of the edge weights between v_i and its neighbors. λ is the scale parameter which depends on the specific type, structure, and properties of the input network. To make the problem simple, the scale parameter will be set as $\lambda = 1$. Based on the assumption that the shape parameter does not force the starting and ending vertices of p, Formula (20.10) is converted into

$$S(p) = w_{0,1} \cdot \prod_{i=1}^{n-1} \frac{w_{i(i+1)}}{d^{\text{wt}}(v_i)}. \tag{20.11}$$

The path strength of a path p thus has a positive relationship with the weights of the edges on p, and a negative relationship with the weighted degrees of the vertices on p. Formula (20.11) also implies that the path strength has an inverse relationship with the length of p because the weighted probability, $w_{i(i=1)}/d^{\text{wt}}(v_i)$, is in the range between 0 and 1, inclusive. As the length of p increases, the product of the weighted probability decreases monotonically. In the same manner, as the average degree of the vertices on p increases, the path strength of p is likely to decrease. For example, in Fig. 20.1, the higher the degrees of the vertices v_1 and v_2, the lower $S(v_0, v_3)$.

The functional influence model considers all possible paths between two vertices including cycles. The quantity of the influence of a vertex on another is defined as the cumulative path strength of all possible paths between them. For example, in Fig. 20.2, suppose we measure the functional influence $F(v_0, v_9)$ of v_0 on v_9 in the weighted network. v_0 becomes the source vertex and v_9 becomes the target vertex. $F(v_0, v_9)$ is calculated by summation of the path strength scores in Formula (20.11) for all possible paths between v_0 and v_9. However, enumerating all possible

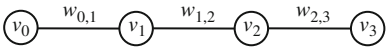


Fig. 20.1 Path strength is determined by three factors: edge weights, weighted degrees of vertices, and path length. To have higher path strength $S(p)$ of the path $p = \langle v_0, v_1, v_2, v_3 \rangle$, the edge weights on p, $w_{0,1}$, $w_{1,2}$, and $w_{2,3}$ should be higher, and the weighted degrees of v_1 and v_2 on p should be lower

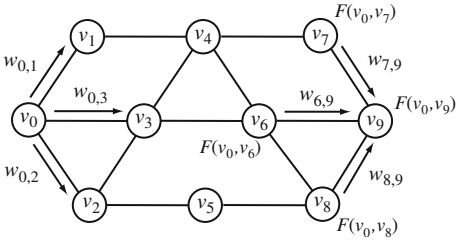


Fig. 20.2 An example of implementing the functional influence model. To calculate the functional influence of v_0 on v_9, $F(v_0, v_9)$, the prior information of the influence of v_0 on the neighbors of v_9, $F(v_0, v_6)$, $F(v_0, v_7)$, and $F(v_0, v_8)$ can be used

paths between two vertices in a large network is not computationally acceptable. An efficient approximation algorithm to simulate functional influences is described in the next section.

20.3.2 Simulation of Functional Influence

The functional influence model has been simulated by a flow-based algorithm. This algorithm has the assumption that flow takes a constant time to traverse each edge. It requires a weighted network and a source vertex s as inputs and produces the distribution of the functional influence of s on all the other vertices in the network as an output.

As the same example in Fig. 20.2, suppose we measure the functional influence, $F(v_0, v_9)$. Among all possible paths between v_0 and v_9, consider the path $p = \langle v_0, v_3, v_6, v_9 \rangle$. By Formula (20.11), the path strength $S(p)$ is considered as $S(\langle v_0, v_3, v_6 \rangle) \times w_{6,9}/d_6^{\text{wt}}$. It indicates that, to compute the functional influence $F(v_0, v_9)$, the prior knowledge of the influence of v_0 on the neighbors of v_9, i.e., $F(v_0, v_6)$, $F(v_0, v_7)$, and $F(v_0, v_8)$, should be obtained in advance. In other words, given $F(v_0, v_6)$, $F(v_0, v_7)$, $F(v_0, v_8)$, the weighted degrees of v_6, v_7, and v_8, and the weights of the edges connecting to v_6, v_7, and v_8, we can estimate $F(v_0, v_9)$. In the same way, $F(v_0, v_6)$, $F(v_0, v_7)$, and $F(v_0, v_8)$ require the prior knowledge of the influence of v_0 on the neighbors of v_6, v_7, and v_8, respectively. Thus, the iterative computation of the influence of v_0 on the other nodes can finally achieve the quantity of the functional influence of v_0 on v_9 through all links across the network.

As a notation in this flow-based algorithm, $f_s(x \to y)$ denotes the functional influence of s, which travels from a vertex x to a vertex y where x and y are linked directly. The initial influence rate $F(s, s)$ can be a user-specific constant value, e.g., 1. Initially, the flow delivers the initial rate of s to its neighbors x with being reduced by weights.

$$f_{init}(s \to x) = w_{s,x} \times F(s, s), \tag{20.12}$$

where $w_{s,x}$ is the normalized weight of the edge between s and x into the range of $0 \leq w_{s,x} \leq 1$. The quantity of the functional influence of s on x, $F(s, x)$, is then updated by summing all incoming flow to x from its neighbors.

$$F(s, x) = \sum_{u \in N(x)} f_s(u \rightarrow x). \tag{20.13}$$

The summation calculates the new functional influence on x through all possible paths in the network. In the initial flow, the functional influence that x receives only comes from the source s. And then, the influence of s traverses all connecting edges by the formula defined as

$$f_s(x \rightarrow y) = \frac{w_{x,y}}{\sum_{z \in N(x)} w_{x,z}} \times F(s, x), \tag{20.14}$$

where $N(x)$ denotes the set of directly connected neighbor vertices of x. The multiplication of those two terms indicates the meaning of the product in Formula (20.11). During the flow, the quantity of the functional influence of s on each vertex v is repeatedly updated by Formula (20.13), traverses connecting edges by Formula (20.14), and is accumulated into $P_s(v)$. The flow on a path stops if the flow $f_s(x \rightarrow y)$ reaches a user-dependent minimum threshold θ_{flow}. The flow simulation starting from s terminates when there is no more flow in the network. The final $P_s(v)$ then represents the functional influence of s on v. The high-level description of the flow-based functional influence simulation algorithm is shown as following:

1. Initialize $F(s, s)$.
2. Compute initial flow $f_{init}(s \rightarrow x)$ by Formula (20.12) for each x where $e(s, x) \in E$.
3. Compute $F(s, x)$ by Formula (20.13) for each x.
4. Compute flow $f_s(x \rightarrow y)$ by Formula (20.14) for each y where $e(x, y) \in E$.
5. Remove flow $f_s(x \rightarrow y)$ if it is less than a threshold θ_{flow}.
6. Replace x and y with y and z, respectively, where $e(y, z) \in E$, and repeat the steps of 3,4,5, and 6 until there is no more flow in the network.
7. Output cumulative $F(s, v)$ for each $v \in V$.

20.3.3 Efficiency Analysis

Efficiency is one of the major strengths of the flow-based functional influence simulation algorithm. In general, random walk simulation on a graph is manipulated by matrix computation. The product of adjacency matrices for each round of random walks runs in $O(n^3)$. For n rounds, the time complexity grows to $O(n^4)$. However, the flow-based simulation algorithm outperforms the matrix computation because it efficiently chases the flow through only existing links and prunes each flow as soon as it becomes trivial. Unlike other graph-theoretic methods, its runtime is obviously

unrelated to the network diameter because flow traverses through all possible paths including cycles. Since this algorithm uses a threshold to halt the flow as a user-specified criterion, the theoretical upper bound of its runtime is unknown. However, the potential factors that affect time complexity of this algorithm are investigated in Fig. 20.3.

Synthetic input networks have been created by different features, and the average runtime of the flow-based simulation starting from randomly selected 200 source vertices in each network has been monitored. In the first test, the networks were produced by increasing the number of vertices, from 500 to 7000, but retaining the density as 0.002. The density of a network represents the proportion of the number of actual edges to the number of all possible edges. In the next test, the networks were produced by the same change of the number of vertices but a constant average degree of 5. In Fig. 20.3 a, when the density is constant, the runtime increases as adding vertices, because of the squared increase of the number of possible edges for the constant density. However, when the average degree is constant, the runtime is uniform regardless of the network size. In the additional experiment, the runtime with the networks produced by the change of density in the fixed number of vertices as 2000 and in a constant average degree of 5. As shown in Fig. 20.3 b, when the network size is fixed, the runtime increases as the density becomes higher. However, when the average degree is constant, the runtime is also uniform regardless of the network density. These results indicate that the average degree of input networks is a more critical factor to reduce time complexity of the flow-based simulation algorithm than the size or density of networks. Since the average degree of complex biological networks is typically low in a power law degree distribution, the flow-based simulation algorithm efficiently outputs the functional influence between any two vertices in the network.

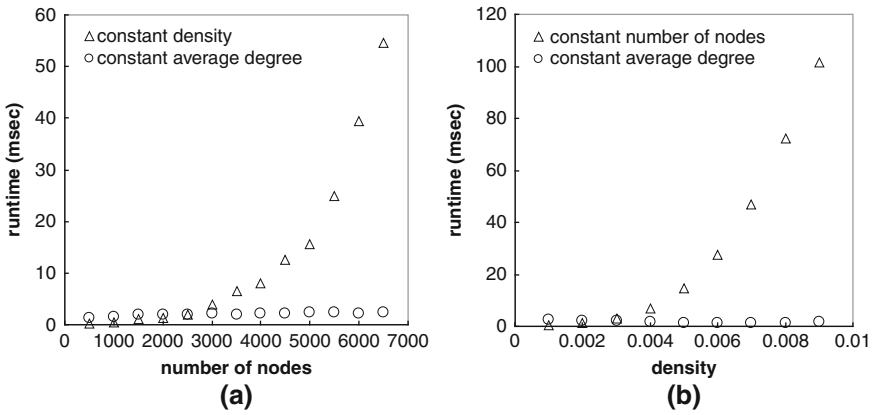


Fig. 20.3 Runtime of flow-based functional influence simulation in synthetic networks. The networks are structured by (**a**) the change of the number of vertices in constant density or a constant average degree and by (**b**) the change of density in the constant number of vertices or a constant average degree

20.4 Modularization Algorithm

The quantity of the functional influence of a source vertex s on a target vertex t can be a decisive indication whether s and t are the members of the same functional module. A potential functional module including s can thus be found by collecting all the vertices under the strong functional influence of s. The modularization algorithm, specific to biological networks, will be presented in this section. The input is a weighted, undirected network, and the output is a collection of functional modules as sets of vertices. The three steps of the functional influence-based modularization algorithm, source selection, functional flow simulation, and post-process, will be discussed in the following subsections.

20.4.1 Source Selection

In this modularization algorithm, the members of a functional module are selected by the functional influence of a source. The source vertex is thus a representative core in the module. The source vertices are chosen through the topological analysis of biological networks, generally via the use of centrality metrics. Commonly used topology-based metrics include degree and clustering coefficient. A previous study [24] has observed that the local connectivity of vertices in biological networks plays a crucial role in cellular functions. It means high-degree vertices are possibly the cores in functional modules. The clustering coefficient in Formula (20.7) also seeks the vertex located in the center of a densely connected region, which is a potential core of a functional module.

In a weighted network, the degree and clustering coefficient can be extended to the weighted degree and weighted clustering coefficient [5]. The weighted degree $d^{\mathrm{wt}}(v_i)$ of a vertex v_i is the summation of the weights between v_i and its neighbors.

$$d^{\mathrm{wt}}(v_i) = \sum_{v_j \in N(v_i)} w_{ij}, \tag{20.15}$$

where w_{ij} is the weight of the edge between v_i and v_j. The weighted clustering coefficient $c^{\mathrm{wt}}(v_i)$ of a vertex v_i is defined as

$$c^{\mathrm{wt}}(v_i) = \frac{1}{d^{\mathrm{wt}}(v_i)(d(v_i)-1)} \sum_{\substack{v_j, v_k \in N(v_i), \\ \langle v_j, v_k \rangle \in E}} \frac{(w_{ij} + w_{ik})}{2}, \tag{20.16}$$

where $d(v_i)$ is the (unweighted) degree of v_i. Then the vertices with high weighted degrees or high weighted clustering coefficients are good candidates as sources. Because the weights are obtained from functional knowledge in general, the weighted degree and weighted clustering coefficient of a vertex include the factors related to not only topological significance in the network but also biological

essentiality. The number of selected source vertices is a user-dependent parameter in this algorithm.

20.4.2 Functional Influence Simulation

The functional influence simulation algorithm has been discussed in the previous section. By flow-based simulation starting from each source vertex, the vertices, on which have a higher functional influence than the minimum influence threshold θ_{inf}, are grouped into a functional module. The flow-based simulation starting from all source vertices then achieves the set of modules, called preliminary modules. Those preliminary modules are typically overlapped because a vertex may be under the functional influence of two or more source vertices.

20.4.3 Post-process

The quality of preliminary modules depends on the proper selection of source vertices. If a selected source vertex is not a core of the potential module, the result will be inaccurate. To improve the accuracy, merging similar preliminary modules or finding optimal source vertices is necessary as a post-processing step.

20.4.3.1 Merging Similar Modules

Two or more preliminary modules may be very similar, i.e., they may include a large fraction of common members, when their source vertices are closely located to each other in the network. The close connection between two sources leads to high functional similarity. Merging such similar preliminary modules is an important step to make final modules accurate. The similarity $S(M_s, M_t)$ between two modules M_s and M_t is measured by the weighted inter-connectivity defined as

$$S(M_s, M_t) = \frac{\sum_{x \in M_s, y \in M_t} c(x, y)}{\min(|M_s|, |M_t|)}, \tag{20.17}$$

where

$$c(x, y) = \begin{cases} 1 & \text{if } x = y \\ w(x, y) & \text{if } x \neq y \text{ and } \langle x, y \rangle \in E \\ 0 & \text{otherwise.} \end{cases} \tag{20.18}$$

The modules with the highest similarity in Formula (20.17) are iteratively merged until the highest similarity is less than a user-specified merging threshold.

20.4.3.2 Iterative Centroid Search

If a vertex on periphery of a real functional module is chosen as a source vertex, then the output modules by the functional influence simulation algorithm would not be functionally homogeneous. The iterative centroid search (ICES) algorithm has been developed to delineate the optimal positions for sources and to precisely identify functional modules [12]. It computes the centrality $C(v_i)$ of a vertex v_i as the sum of the maximum path strengths from v_i to the other vertices in the network.

$$C(v_i) = \sum_{\substack{v_j \in V, \\ i \neq j}} S_{\max}(\langle v_i, \ldots, v_j \rangle), \tag{20.19}$$

The centrality measurement guides the selection of a centroid in each module generated by flow-based simulation. The vertex with the highest centrality in each preliminary module becomes a new centroid of the module. These centroids then become the source vertices for the next round of flow-based simulation.

The ICES algorithm iterates two procedures: the selection of a centroid in each module and the flow-based simulation starting from each new centroid to generate a set of modules. Each iterative step identifies a set of centroids progressively closer to the actual cores of functional modules. If an initial centroid is located on the periphery of a potential module, the centroid approaches the actual core of the module during the iterations. The algorithm concludes by optimizing the starting positions of flow-based simulation, thus identifying the most accurate functional modules.

20.5 Application to Protein Interaction Networks

Protein interaction networks are structured by protein–protein interaction data. They are represented by an undirected, unweighted graph $G(V, E)$ with proteins as a set of vertices V and the interactions between them as a set of edges E. An example of the yeast protein interaction network is illustrated in Fig. 20.4. Currently, protein–protein interaction data are publicly available in several databases such as BioGRID [6], MIPS [27], DIP [34], MINT [10], and IntAct [25]. Since proteins interact with each other for biochemical stability and functionality, the interaction evidence between proteins can be interpreted as their functional coherence, and functional modules of proteins can be identified based on the connectivity in the interaction networks.

20.5.1 Data Source

The functional influence-based approach has been tested in the yeast protein interaction network. The core protein–protein interaction data have been extracted from DIP [34]. They include 2526 distinct proteins and 5949 interactions between them.

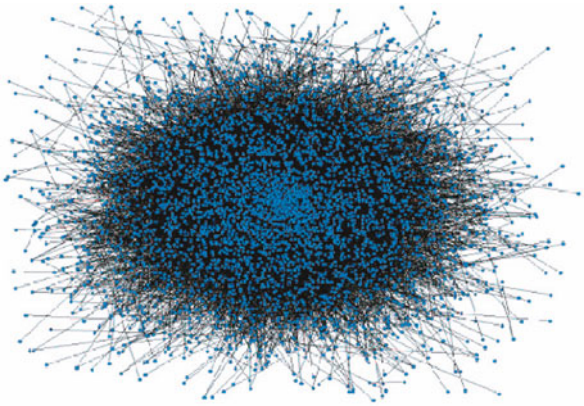


Fig. 20.4 The yeast protein interaction network. It is represented as a large-scale, complex graph

Since the functional influence-based approach requires the weighted graph as an input, the weights of the interactions have been computed using the annotation data from gene ontology (GO) [19] which currently has the most comprehensive functional information of molecules. The GO is a repository of semantic terms and their relationships across organisms. The GO terms are structured as a directed acyclic graph according to the "is-a" and "part-of" relationships among them. Characterized genes or proteins are annotated on each term. Using GO data, semantic similarity and semantic interactivity [11] between interacting proteins have been measured to estimate functional similarity between them. A semantic similarity or semantic interactivity score of each interaction is assigned to the corresponding edge as a weight.

The semantic similarity is assessed by the information contents in two semantic concepts. In information theory, self-information is a measure of the information content associated with the outcome of a random variable. The amount of self-information contained in an event c depends on the probability $P(c)$ of the event. More specifically, the smaller the probability of the event, the larger the self-information to be received when the event indeed occurs. The information content of a concept C in the taxonomy is then defined as the negative log likelihood of C, $-\log P(C)$. Semantic similarity between two concepts is measured by their commonality, i.e., more common information two concepts share, more similar they are. Resnik [32] proposed to measure the semantic similarity of the concepts, C_1 and C_2, by the information content of the most specific concept C_0 that subsumes both C_1 and C_2.

The semantic similarity between interacting proteins can be derived from the commonality of information contents of two GO terms having the annotations of two interacting proteins. Suppose the size of annotation represents the number of annotated proteins on a GO term. Using the annotation size of the most specific GO term, on which two proteins x and y are annotated, semantic similarity $S_{\text{sem}}(x, y)$ between x and y is defined as

$$S_{\text{sem}}(x, y) = -\log\left(\min_i P_i(x, y)\right), \tag{20.20}$$

where $P_i(x, y)$ is the ratio of the number of proteins annotated on the GO term t_i, whose annotation includes both x and y, to the total number of distinct proteins in annotations. Since x and y can be annotated on several different GO terms, the minimum $P_i(x, y)$ is chosen to maximize the similarity.

The semantic interactivity is based on the connection patterns among proteins annotated on specific GO terms. It measures the probability $P(x, y)$ that x interacts with the proteins annotated on the GO terms whose annotation includes y as

$$P(x, y) = \frac{\max_i |S_i(y) \cap N'(x)|}{|N'(x)|}, \tag{20.21}$$

where $N'(x) = N(x) \cup \{x\}$. If x and all of its neighbors are not included in $S_i(y)$ for any i, then $P(x, y)$ is 0. If all of them are included in a set $S_i(y)$, then $P(x, y)$ is 1. Equation (20.21) thus satisfies the range of $0 \leq P(x, y) \leq 1$. The semantic interactivity $I_{\text{sem}}(x, y)$ between x and y is then measured by the geometric mean of $P(x, y)$ and $P(y, x)$.

$$I_{\text{sem}}(x, y) = \sqrt{P(x, y) \times P(y, x)}. \tag{20.22}$$

20.5.2 Identification of Overlapping Modules

The functional influence-based modularization algorithm requires two important user-dependent parameters: the number of sources and the minimum influence threshold. The number of modules in an output set depends on the number of sources. On the other hand, the minimum influence threshold determines the average size of output modules. By alternating the two parameter values, the optimal output sets of modules have been found. Since real functional modules are hierarchically distributed in typical, the output sets having similar numbers of the real functional categories on the first, second, and third level in a hierarchy from the MIPS database [27] are chosen.

The output modules share a large number of common members. The overlapping rates of output modules have been evaluated by counting the number of appearance across different modules for each protein. In Table 20.1, the average overlapping rates on the output sets of modules are compared with that on real functional categories from MIPS. As the protein interaction network is decomposed into more specific functional modules in a lower level, the average overlapping rate slightly increases. The output modules weighted by semantic similarity have lower overlapping rates than real functional modules whereas those by semantic interactivity have higher overlapping rates. However, the increasing patterns of the rates in output modules are similar to those in real modules. Overall, the modules identified by

Table 20.1 Average overlapping rates of proteins in the modules on the first, second, and third level in a functional hierarchy

Module set	Functional categories	Output modules (semantic similarity)	Output modules (semantic interactivity)
Level 1	3.05	2.71	3.84
Level 2	3.45	3.25	3.98
Level 3	3.71	3.42	4.05

this functional influence-based approach have a similar overlapping pattern when compared to real functional categories.

20.5.3 Statistical Assessment of Modules

To statistically assess the identified modules, the p-value from the hypergeometric distribution [9, 11] is commonly used. Each module is mapped to a reference function with the lowest p-value and evaluated by the negative of $\log(p\text{-value})$. A low p-value (equivalently, a high $-\log(p\text{-value})$) between an identified module and a reference function indicates that the module closely corresponds to the function. In this experiment, the functional categories and their annotations from the MIPS database were used as the reference functions.

The performance of the functional influence-based modularization algorithm is compared to the competing methods in two different categories: the betweenness-based algorithm [18] as a hierarchical approach and the clique percolation algorithm [30] as a density-based method. For each implementation, the parameter values that resulted in the best accuracy have been selected. Table 20.2 shows the parameter values and the results of the output modules. Although the clique percolation method is able to find overlapping modules, it detects numerous small-sized modules with a few disproportionally large modules. As a result, its average accuracy is lower than the other methods. The betweenness-based hierarchical clustering method includes most of the sparsely connected vertices in the output modules. However, because the output modules are disjoint, this method has a lower accuracy than the functional influence-based method. In summary, the functional influence-based algorithm outperforms other competing methods in terms of the accuracy of functional module identification in a real protein interaction network.

Table 20.2 Performance comparison of modularization methods. The output modules were statistically assessed in terms of accuracy using p-value

Method	Number of modules	Average size of modules	$-\log(p\text{-value})$
Influence-based (semantic similarity)	178	46.14	24.42
Influence-based (semantic interactivity)	189	40.40	29.05
Betweenness-based	57	41.02	17.44
Clique percolation	57	17.86	12.32

20.6 Conclusion

Discovering hidden knowledge related to molecular functions from complex biological networks has been a crucial task in biological data mining because the system-level understanding of functional behaviors of molecular components becomes the underlying bases of practical biomedical applications. Interestingly, it has been observed that biological networks have collective behaviors in topology. In this chapter, a novel functional influence-based approach has been proposed for effective analysis of complex biological networks. This approach is based on the functional influence model and flow-based simulation algorithm that quantifies the functional influence of each vertex on the others through links. This influence pattern becomes a unique characteristic of the input network in terms of connectivity and functionality. The functional influence simulation culminates in identifying functional modules hidden in the complex biological network. This framework will be enhanced by applying top-notch techniques to the source seed selection as a pre-process and the cluster merging as a post-process. Moreover, a better edge-weighting scheme, which integrates with heterogeneous biological data available, can be adopted for higher accuracy in detecting clusters corresponding to real functional modules.

Acknowledgments This work was partly supported by NSF grant DBI-0234895 and NIH grant I P20 GM067650-01A1.

References

1. M. Altaf-Ul-Amin, Y. Shinbo, K. Mihara, K. Kurokawa, and S. Kanaya. Development and implementation of an algorithm for detection of protein complexes in large interaction networks. *BMC Bioinformatics*, 7: 207, 2006.
2. V. Arnau, S. Mars, and I. Marin. Iterative cluster analysis of protein interaction data. *Bioinformatics*, 21: 364–378, 2005.
3. G. D. Bader, and C. W. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4: 2, 2003.
4. A.-L. Barabasi, and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews: Genetics*, 5, 101–113, 2004.
5. A. Barrat, M. Barthelemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. *Proceedings of the National Academy of Science USA*, 101, 3747–3752, 2004.
6. B.-J. Breitkreutz, C. Stark, T. Reguly, L. Boucher, A. Breitkreutz, M. Livstone, R. Oughtred, D. H. Lackner, J. Bahler, V. Wood, K. Dolinski, and M. Tyers. The BioGRID interaction database: 2008 update. *Nucleic Acids Research*, 36, D637–D640, 2008.
7. S. Brohee, and J. van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7, 488, 2006.
8. C. Brun, C. Herrmann, and A. Guenoche. Clustering proteins from interaction networks for the prediction of cellular functions. *BMC Bioinformatics*, 5, 95, 2004.
9. D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acid Research*, 31, 2443–2450, 2003.

10. A. Chatr-aryamontri, A. Ceol, L. Montecchi-Palazzi, G. Nardelli, M. V. Schneider, L. Castagnoli, and G. Cesareni. MINT: the Molecular INTeraction database. *Nucleic Acid Research*, 35, D572–D574, 2007.
11. Y.-R. Cho, W. Hwang, M. Ramanathan, and A. Zhang. Semantic integration to identify overlapping functional modules in protein interaction networks. *BMC Bioinformatics*, 8, 265, 2007.
12. Y.-R. Cho, W. Hwang, and A. Zhang. Optimizing flow-based modularization by iterative centroid search in protein interaction networks. In *Proceedings of 7th IEEE Symposium on Bioinformatics and Bioengineering (BIBE)*, pages 342–349, 2007.
13. Y.-R. Cho, L. Shi, and A. Zhang. flowNet: Flow-based approach for efficient analysis of complex biological networks. In *Proceedings of 9th IEEE International Conference on Data Mining (ICDM)*, pages 91–100, 2009.
14. I. Derenyi, G. Palla, and T. Vicsek. Clique percolation in random networks. *Physical Review Letters*, 94, 160202, 2005.
15. S. Van Dongen. A new clustering algorithm for graphs. Technical Report, National Research Institute for Mathematics and Computer Science in the Netherlands, INS-R0010, 2000.
16. R. Dunn, F. Dudbridge, and C. M. Sanderson. The use of edge-betweenness clustering to investigate biological function in protein interaction networks. *BMC Bioinformatics*, 6, 39, 2005.
17. A. J. Enright, S. van Dongen, and C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research*, 30, 1575–1584, 2002.
18. M. Girvan, and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Science USA*, 99, 7821–7826, 2002.
19. The Gene Ontology Consortium. The Gene Ontology project in 2008. *Nucleic Acids Research*, 36, D440–D444, 2008.
20. D. S. Goldberg, and F. P. Roth. Assessing experimentally derived interactions in a small world. *Proceedings of the National Academy of Science USA*, 100, 4372–4376, 2003.
21. E. Hartuv, and R. Shamir. A clustering algorithm based on graph connectivity. Information Processing Letters, 76, 175–181, 2000.
22. L. H. Hartwell, J. J. Hopfield, S. Leibler, and A. W. Murray. From molecular to modular cell biology. *Nature*, 402, c47–c52, 1999.
23. P. Holme, M. Huss, and H. Jeong. Subnetwork hierarchies of biochemical pathways. *Bioinformatics*, 19, 532–538, 2003.
24. H. Jeong, S. P. Mason, A.-L. Barabasi, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411, 41–42, 2001.
25. S. Kerrien, et al. IntAct—open source resource for molecular interaction data. *Nucleic Acids Research*, 35, D561–D565, 2007.
26. A. D. King, N. Przulj, and I. Jurisica. Protein complex prediction via cost-based clustering. *Bioinformatics*, 20, 3013–3020, 2004.
27. H. W. Mewes, S. Dietmann, D. Frishman, R. Gregory, G. Mannhaupt, K. F. X. Mayer, M. Munsterkotter, A. Ruepp, M. Spannagl, V. Stumptflen, and T. Rattei. MIPS: Analysis and annotation of genome information in 2007. *Nucleic Acid Research*, 36, D196–D201, 2008.
28. M. E. J. Newman. Scientific collaboration networks. II. Shortest paths, weighted networks and centrality. *Physical Review E*, 64, 016132, 2001.
29. M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, 066133, 2004.
30. G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435, 814–818, 2005.
31. F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Science USA*, 101, 2658–2663, 2004.
32. P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of 14th International Joint Conference on Artificial Intelligence*, pages 448–453, 1995

33. A. W. Rives, and T. Galitski. Modular organization of cellular networks. *Proceedings of the National Academy of Science USA*, 100, 1128–1133, 2003.
34. L. Salwinski, C. S. Miller, A. J. Smith, F. K. Pettit, J. U. Bowie, and D. Eisenberg. The database of interacting proteins: 2004 update. *Nucleic Acid Research*, 32, D449–D451, 2004.
35. M. P. Samanta, and S. Liang. Predicting protein functions from redundancies in large-scale protein interaction networks. *Proceedings of the National Academy of Science USA*, 100, 12579–12583, 2003.
36. R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Molecular Systems Biology*, 3, 88, 2007.
37. V. Spirin, and L. A. Mirny. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Science USA*, 100, 12123–12128, 2003.
38. I. V. Tetko, A. Facius, A. Ruepp, and H. W. Mewes. Super paramagnetic clustering of protein sequences. *BMC Bioinformatics*, 6, 82, 2005.
39. Z. Wang, and J. Zhang. In search of the biological significance of modular structures in protein networks. *PLoS Computational Biology*, 3, e107, 2007.
40. D. J. Watts, and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393, 440–442, 1998.
41. S. Wuchty, and E. Almaas. Peeling the yeast protein network. *Proteomics*, 5, 444–449, 2005.
42. A. Zhang. *Protein Interaction Networks: Computational Analysis*. Cambridge University Press New York, NY, 2009.
43. E. Zotenko, K. S. Guimaraes, R. Jothi, and T. M. Przytycka. Decomposition of overlapping protein complexes: a graph theoretical method for analyzing static and dynamic protein associations. *Algorithms for Molecular Biology*, 1, 7, 2006.

Chapter 21
Gene Reachability Using Page Ranking on Gene Co-expression Networks

Pinaki Sarder, Weixiong Zhang, J. Perren Cobb, and Arye Nehorai

Abstract We modify the Google Page-Rank algorithm, which is primarily used for ranking web pages, to analyze the gene reachability in complex gene co-expression networks. Our modification is based on the average connections per gene. We propose a new method to compute the metric of average connections per gene, inspired by the Page-Rank algorithm. We calculate this average as eight for human genome data and three to seven for yeast genome data. Our algorithm provides clustering of genes. The proposed analogy between web pages and genes may offer a new way to interpret gene networks.

21.1 Introduction

The Page-Rank algorithm [1] analyzes links and assigns a numerical weighting to each element of a hyperlinked set of elements, such as web pages, relative to its importance. The level of importance represents the likelihood that a surfer, surfing the web, will arrive at any stochastic page, based on the page similarity, having started from another page. The Page-Rank theory assumes that a surfer continues clicking the web links with a damping factor probability d [1].

In this chapter, we modify the Page-Rank algorithm and apply it to rank genes in co-expression (CoE) networks. The modification considers the fact that every gene is connected to an average number of neighbors in the CoE network. In [2] it is shown that CoE networks which are based on the average connections per gene are biologically meaningful. Namely, such networks can capture biologically relevant features of the ground-truth gene regulatory networks and can preserve the functional relationships between the genes. The average number of connections per neighbor was applied to analyze ranking of social networks, but apparently not

A. Nehorai (✉)
Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO 63130, USA
e-mail: nehorai@ese.wustl.edu

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8_21,

applied to genes or web pages using the Page-Rank algorithm [2]. In particular, the authors of [3] applied the Page-Rank algorithm to rank genes using conventional CoE networks. We propose a new method to compute the metric of average connections per gene, inspired by the Page-Rank algorithm.

We use the following major steps to implement our algorithm: (i) compute all possible gene-CoE networks from a microarray data set using a nearest-neighbor-based method [2] corresponding to the number of all possible nearest neighbors; (ii) estimate the average number of gene connections per gene using the networks computed in the first step; (iii) recompute the CoE network based on the average gene connections; (iv) apply the Page-Rank algorithm to rank genes using the CoE network as computed in the third step; (v) Define molecular and functional classifications of the top-ranked genes using the popular knowledge-based software Ingenuity Pathway Analysis (IPA) (http://www.ingenuity.com).

We show through examples that our algorithm provides clustering of gene data. We applied our algorithm to real data sets on ventilator-associated pneumonia (VAP) [4] and ExpressDB (http://arep.med.harrard.edu/ExpressDB/yeastindex.html) of yeast gene expressions. The pathways formed by the top-ranked VAP genes are well described in infection/pneumonia literature (http://www.ingenuity.com), adding a level of verification to our propositions. We calculated the average number of gene connections per gene and found them to be eight for human and three to seven for yeast genome data.

21.2 Method

In this section, we describe our method for ranking genes. We first present our construction of the CoE networks based on the nearest neighbors. Then we discuss the Page-Rank algorithm, and its modification for ranking genes using the CoE networks based on nearest neighbors.

21.2.1 Nearest-Neighbor-Based CoE Networks

We start with the notations that we use for constructing the CoE networks. Denote the expression of the jth gene of the kth patient at the lth time point as $x_{kj}(l)$, where $j \in \{1, 2, \ldots, G\}$, $k \in \{1, 2, \ldots, K\}$, and $l \in \{1, 2, \ldots, N\}$. Let the gene expression vector be $\boldsymbol{x}_{kj} = [x_{kj}(1), x_{kj}(2), \ldots, x_{kj}(N)]^{\mathrm{T}}$, where "T" denotes the vector transpose. Define the average expression of the jth gene over the K patients as

$$\boldsymbol{x}_{\cdot j} = (1/K) \sum_{k=1}^{K} \boldsymbol{x}_{kj}, \tag{21.1}$$

and the Pearson correlation coefficient [2] between the ith and jth genes as $\boldsymbol{C}(i, j) = \text{corr}(\boldsymbol{x}_{.i}, \boldsymbol{x}_{.j})$, where "corr" denotes the correlation operation. Note that the symmetric correlation matrix is $\boldsymbol{C}$ of size $G \times G$.

We present our construction of the CoE networks based on the nearest neighbors. For every ith gene, we sort all other genes, j ($\forall\ j \neq i$), in a descending order by their correlation coefficients to the ith gene. Denote the sorted order of the jth gene for the ith gene using an asymmetric metric $\boldsymbol{R}(i, j)$ ($\forall\ \boldsymbol{R}(i, j) \in \mathbb{N}$). We define $\boldsymbol{R}(i, i) = 0$ since we sort the correlation coefficients of all genes, j, with the ith gene ($\forall\ j \neq i$). We connect every ith gene ($i \in \{1, 2, \ldots, G\}$), starting from its top α co-expressed genes (nearest neighbors) using the sorted order, where α is a user-defined threshold. The sorted order of the jth gene with respect to the ith gene ($\forall\ i \neq j$), $\boldsymbol{R}(i, j)$, is in general not equal to $\boldsymbol{R}(j, i)$ ($\forall\ j \neq i$). Some nodes may have more than α edges, due to the asymmetric property of the ranking. That is, even though a specific ith ($i \in \{1, 2, \ldots, G\}$) gene lists only α genes as its friends, other genes that are not on i's friend-list may have i as their friends [2]. Using this method, we define gene-CoE matrices $\boldsymbol{W}_{\text{l}}$ and $\boldsymbol{W}_{\text{c}}$ as,

$$\boldsymbol{W}_{\text{l}}(i, j) = \begin{cases} 1 & \text{if } \boldsymbol{R}(i, j) \leq \alpha \\ 0 & \text{otherwise} \end{cases}, \tag{21.2}$$

and

$$\boldsymbol{W}_{\text{c}}(i, j) = \begin{cases} \frac{\boldsymbol{C}(i,j)+1}{2} & \text{if } \boldsymbol{R}(i, j) \leq \alpha \\ 0 & \text{otherwise} \end{cases}, \tag{21.3}$$

where $\boldsymbol{W}_{\text{l}}(i, j)$ and $\boldsymbol{W}_{\text{c}}(i, j)$ are the $(i, j)^{\text{th}}$ elements of the $G \times G$ dimensional CoE matrices $\boldsymbol{W}_{\text{l}}$ and $\boldsymbol{W}_{\text{c}}$, respectively. Here we obtain CoE networks of different granularities by varying α in (21.2) and (21.3), as all nodes in these networks are connected [2]. Also, we define (21.2) and (21.3) such that, $0 \leq \boldsymbol{W}_{\text{l}}(i, j), \boldsymbol{W}_{\text{c}}(i, j) \leq 1$. The lower limit is required for employing the Page-Rank method. The upper limit is chosen for the purpose of normalization.

We propose to use the matrix $\boldsymbol{W}_{\text{l}}$ or $\boldsymbol{W}_{\text{c}}$ for ranking genes. Note that due to the asymmetric property of neighbors, $\boldsymbol{W}_{\text{l}}$ and $\boldsymbol{W}_{\text{c}}$ construct directional networks. Hence, a non-zero $\boldsymbol{W}_{\text{l}}(i, j)$ or $\boldsymbol{W}_{\text{c}}(i, j)$ signifies the reachability from the jth gene to the ith gene [2].

The CoE networks (21.2) and (21.3) can capture biologically relevant features of the ground-truth gene regulatory networks and can preserve the functional relationships between the genes. Namely, the authors in [2] found that in such CoE networks the genes that are in the same pathway or functional complex tend to be close to one another with direct links or short paths. They also found that clustering of such networks can produce biologically more meaningful results than directly clustering the gene expression data with conventional clustering methods. They further found that clustering of such CoE networks is robust; particularly, perturbing a large fraction of the connections of such networks does not significantly affect their final clustering results.

21.2.2 Page-Rank Method for Ranking Genes

Using analogy between the Internet web and gene network, the Page-Rank applied to the CoE network provides the likelihood that a gene is reachable by connecting edges. The connection path starts from any other gene and proceeds to the target gene by connecting based on the expression similarity. Similarly, d, in CoE networks means the probability of propagating the signal from one gene to the another. Generally, the value of d is set near to 1 (around 0.85) [1, 3] for exploiting the connectivity information in gene ranking; we employ $d = 0.85$ in our gene-rank analysis.

Denote the Page-Rank scores of genes $1, 2, \ldots, G$ as $\Pr(1), \Pr(2), \ldots, \Pr(G)$, where $\Pr(i)$ is the probability of reaching the ith gene by following the connecting edges starting from any other gene based on the expression similarity. Define

$$\Pr(i) = \frac{1-d}{G} + d \sum_{j \in M(i)} \boldsymbol{W}(i, j) \Pr(j), \tag{21.4}$$

where G is the total number of genes in the network, $M(i)$ denotes the genes connected to the ith gene, and $\boldsymbol{W}(i, j)$ is

$$\boldsymbol{W}(i, j) = \begin{cases} \left(\boldsymbol{W}_1(i, j) / \sum_{i=1}^{G} \boldsymbol{W}_1(i, j) \right) & \text{or} \\ \left(\boldsymbol{W}_c(i, j) / \sum_{i=1}^{G} \boldsymbol{W}_c(i, j) \right), \end{cases} \tag{21.5}$$

where $\boldsymbol{W}(i, j)$ is the $(i, j)^{\text{th}}$ element of the matrix $\boldsymbol{W}$ of size $G \times G$ [1]. Here $\Pr(i)$ is proportional to $\boldsymbol{W}(i, j)$. The value of $\boldsymbol{W}(i, j)$ is 0 if the jth gene does not link to the ith gene, and $\boldsymbol{W}(i, j)$ is normalized such that for each j, $\sum_{i=1}^{G} \boldsymbol{W}(i, j) = 1$; i.e., the elements of each column of $\boldsymbol{W}$ add up to one. We define $\boldsymbol{r} = [\Pr(1), \Pr(2), \ldots, \Pr(G)]^T$. The solution of $\boldsymbol{r}$ is

$$\boldsymbol{r} = \left(\frac{1-d}{G} \right) (\boldsymbol{I} - d\boldsymbol{W})^{-1} \boldsymbol{1}, \tag{21.6}$$

where $\boldsymbol{1}$ is a $G \times 1$ vector of ones and $\boldsymbol{I}$ is an identity matrix. We rank the genes using their corresponding scores.

Gene ranking by applying the Page-Rank algorithm to gene networks has a strong biological interpretation. For example, suppose a gene with a low expression value is a transcription factor that controls all the genes connected with it. The transcription factor itself may be "activated" in the ground-truth gene regulatory networks, and the genes connected with it show high expression values. The Page-Rank algorithm should be able to identify this transcription factor gene.

In principle, we can rank genes by applying the Page-Rank algorithm to any network that defines the connectivity information among the genes. This is analogous to ranking web pages, where we rank these pages using the Internet web connectivity information among the web pages; see the beginning of this section. In the literature, either Gene Oncology annotations or the CoE network is used for ranking genes [3]. (Gene Oncology annotations provide gene networks based on past biological studies.) In this chapter, we use CoE networks defined by (21.2) and (21.3) for ranking genes. Recall from Section 21.2.1 that these networks are biologically relevant. Thus, gene ranking using the Page-Rank algorithm in our work should provide biologically significant results; see Sections 21.4 and 21.5 for more details.

21.2.3 Replacing α by the Average Number of Connections per Gene

In the gene ranking algorithm described above, we replace α by the average number of connections, α_{m}, per gene in the CoE network. In Section 21.3, we discuss a numerical method for estimating α_{m}.

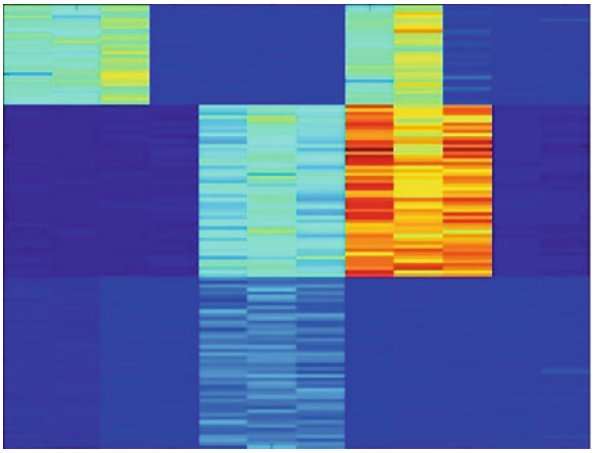
21.3 Numerical Examples

We present numerical examples to investigate: (i) the value of the parameter α_{m} for the CoE matrices $\boldsymbol{W}_{\mathrm{l}}$ and $\boldsymbol{W}_{\mathrm{c}}$ and (ii) the significance of that optimal value for microarray gene-data clustering.

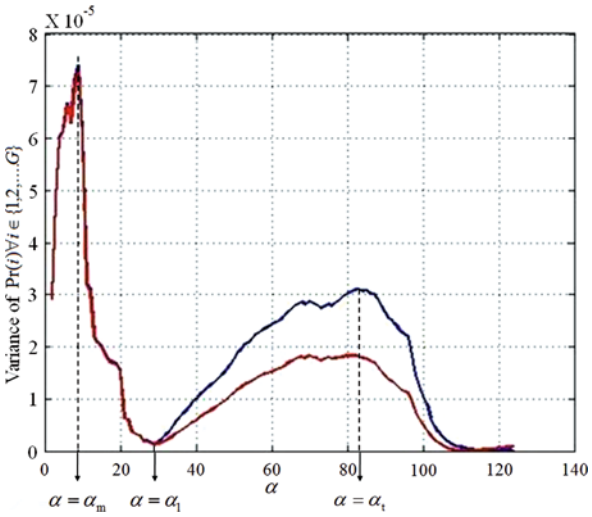
21.3.1 Estimating α_{m}

We simulated three clusters of genes with 12 samples in each using the data generation scheme as proposed in [5]. We computed the gene ranks using $\boldsymbol{W}_{\mathrm{l}}$ and $\boldsymbol{W}_{\mathrm{c}}$ for the simulated data set. We computed the variance of the gene-rank scores across different networks for varying α. This variance signifies how the network connection density changes from one gene to another. Small variance means all the genes in the network are likely to be uniformly reachable with a similar probability. Large variance indicates that some genes are more likely to be reachable than the others. In general, a few genes of an organism are responsive to a specific perturbation. Hence, we investigate the parameter α that leads to the maximum variance in the corresponding gene-rank scores (see Fig. 21.1a,b). We observe in Fig. 21.1b that the gene-rank score variance behaves similarly using the matrix $\boldsymbol{W}_{\mathrm{l}}$ or $\boldsymbol{W}_{\mathrm{c}}$.

We propose to compute the *average number of connections per gene* for any of the CoE matrices $\boldsymbol{W}_{\mathrm{l}}$ and $\boldsymbol{W}_{\mathrm{c}}$ as



(a)



(b)

Fig. 21.1 (**a**) Heatmap of the simulated gene data set with three clusters. (**b**) Variance of $\Pr(i)$ $(\forall\, i \in \{1, 2, \ldots, G\})$ as a function of α. The *warm grey* and *cool grey curves* show the results using the gene–CoE matrices $\boldsymbol{W}_{\mathrm{c}}$ and $\boldsymbol{W}_{\mathrm{l}}$, respectively. We indicate the local maxima using $\alpha = \alpha_{\mathrm{m}}$ and $\alpha = \alpha_{\mathrm{t}}$ and the local minimum using $\alpha = \alpha_{\mathrm{l}}$

$$\alpha_{\mathrm{m}} = \underset{\alpha}{\operatorname{argmax}}\ \mathrm{Var}\{\Pr(i) : i \in \{1, 2, \ldots, G\}|\alpha\}, \tag{21.7}$$

where $\mathrm{Var}\{\cdot\}$ is the variance of the elements in $\{\cdot\}$ and $\{\Pr(i) : i \in \{1, 2, \ldots, G\}|\alpha\}$ denotes the Page-Rank scores of the genes $1, 2, \ldots, G$ for a given α. We compute (21.7) for all possible values of α to obtain α_{m}. The CoE matrix generated using $\alpha = \alpha_{\mathrm{m}}$ is the most diverse with respect to the connection density from one gene to another.

21.3.2 Clustering

We relate our proposed gene-rank analysis, using high d value, with gene-data clustering. Consider (see Fig. 21.1b):

1. $\alpha = \alpha_{\mathrm{m}}$ where $\mathrm{Var}\{\Pr(i) : i \in \{1, 2, \ldots, G\}|\alpha\}$ attains a global maximum,
2. $\alpha = \alpha_{\mathrm{l}}$ where $\mathrm{Var}\{\Pr(i) : i \in \{1, 2, \ldots, G\}|\alpha\}$ attains a local minimum,
3. $\alpha = \alpha_{\mathrm{t}}$ where $\mathrm{Var}\{\Pr(i) : i \in \{1, 2, \ldots, G\}|\alpha\}$ attains a local maximum.

Figure 21.2a–c shows the corresponding histograms for $\alpha = \alpha_{\mathrm{m}}$, $\alpha = \alpha_{\mathrm{l}}$, and $\alpha = \alpha_{\mathrm{t}}$, respectively. The histogram of $\{\Pr(i) : i \in \{1, 2, \ldots, G\}|\alpha\}$ is unimodal (single large peak) for $\alpha \leq \alpha_{\mathrm{l}}$ and bimodal (two large peaks) for $\alpha > \alpha_{\mathrm{l}}$. Figure 21.2d–f shows the heatmaps of the corresponding CoE matrix $\boldsymbol{W}_{\mathrm{l}}$ for $\alpha = \alpha_{\mathrm{m}}$, $\alpha = \alpha_{\mathrm{l}}$, and $\alpha = \alpha_{\mathrm{t}}$, respectively. We find here that for $\alpha \leq \alpha_{\mathrm{l}}$, $\boldsymbol{W}_{\mathrm{l}}$ is block diagonal and classifies accurately the clusters of the data set shown in Fig. 21.1a. Note that the simulated data that we use here in Fig. 21.1b are arranged cluster by cluster. This is not the case in practice, where an additional permutation step is required for obtaining the block-diagonal structure of $\boldsymbol{W}_{\mathrm{l}}$ for $\alpha \leq \alpha_{\mathrm{l}}$. Note also that the local minimum α_{l} between α_{m} and α_{t} in Fig. 21.1b appears as a result of our numerical simulation, and we do not investigate in this chapter the analytical or intuitive justification of this result. We will investigate this issue in our future work.

Our result here also intuitively suggests a close relationship between gene ranking using the Page-Rank algorithm and gene-data clustering. Namely, genes with similar expression levels that are measured using microarrays are highly correlated with each other. The Page-Rank algorithm finds these genes equally reachable by starting from another gene and thus clusters them together based on their degree of reachability.

21.4 Application to Real Data

We applied our algorithm to real data sets of gene expression to (i) rank genes, (ii) select the most vulnerable genes under possible perturbations, and (iii) estimate the average number of connections per gene in the respective CoE networks.

21.4.1 Data Sets

Our real data analysis involves VAP [4] and ExpressDB (http://arep.med.harvard.edu/ExpressDB/yeastindex.html) yeast mRNA expression data sets. We briefly describe these data sets below.

VAP expression profiles were generated from 27 patients every two days for up to 3 weeks [4]. A 7-day VAP window was defined bracketing the clinical diagnosis of pneumonia, with day 0 being the day that a patient was diagnosed as

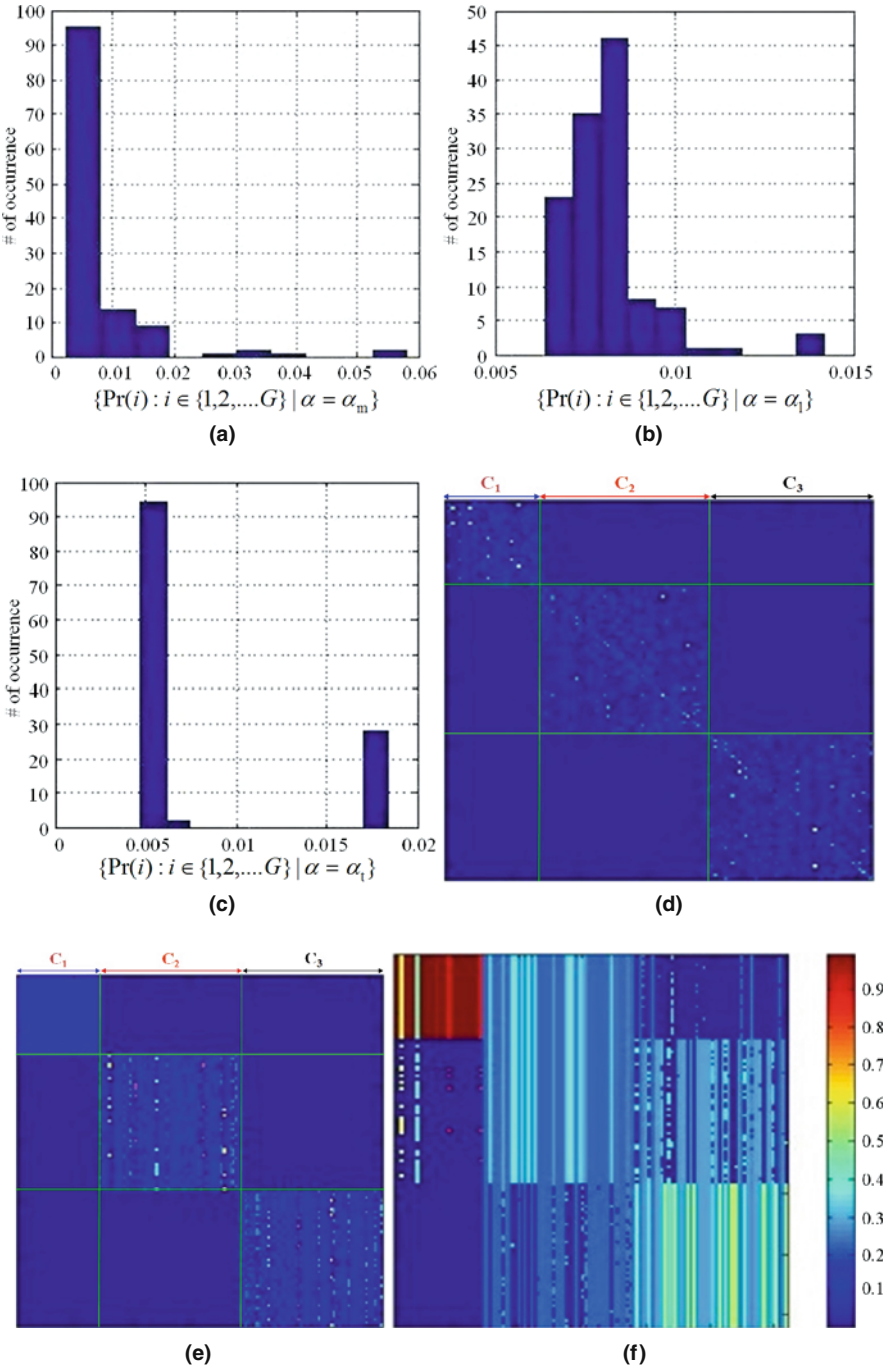


Fig. 21.2 Histogram of $\{\Pr(i) : i \in \{1, 2, \ldots, G\}|\alpha\}$ for (**a**) $\alpha = \alpha_{\mathrm{m}}$, (**b**) $\alpha = \alpha_{\mathrm{l}}$, and (**c**) $\alpha = \alpha_{\mathrm{t}}$. Heatmaps of W_{l} for (**d**) $\alpha = \alpha_{\mathrm{m}}$, (**e**) $\alpha = \alpha_{\mathrm{l}}$, and (**f**) $\alpha = \alpha_{\mathrm{t}}$

having VAP. The time points within this VAP window were −3, −1, 0, 1, and 3 days [4]. Using Extraction of Differential Gene Expression (EDGE) software (http://www.edge.software.com/), 85 mRNA species were identified with an FDR of 0.1 whose abundance changed concordantly among the patients during the VAP window [4]. We employed our proposed method to this data set for (i) estimating the average number of connections per gene in the CoE networks and (ii) computing gene rank to reveal additional functionally important genes with weak differential expression.

ExpressDB is an yeast RNA expression data sets for multiple experimental conditions (http://arep.med.harvard.edu/ExpressDB/yeastindex.html). In this data set, our proposed method selected the most vulnerable genes under possible perturbations.

21.4.2 VAP Gene Ranking

We compared the networks based on the IPA (http://www.ingenuity.com/) and Pearson correlation coefficient [2] using their respective gene ranks. We observed that these two networks have somewhat different gene ranks. We found only five (BTK, PIM1, IL1B, ITGA2B, and FLT1) in common among the 20 top-ranked genes in these two networks. Note that the IPA gives an undirectional network based on the reported gene–gene interactions (http://www.ingenuity.com/). For such networks, we redefine (21.5) by assigning

$$\boldsymbol{W}_1(i,\, j) = \begin{cases} 1 & \text{if the corresponding genes are connected, and} \\ 0 & \text{otherwise.} \end{cases} \tag{21.8}$$

21.4.3 Reachability in ExpressDB Yeast RNA Expression Data sets

We claim that our method essentially computes a reachability of a gene from any other gene using a revised Markov system. We employed our method to select the most vulnerable genes to various perturbations in 31 different ExpressDB data sets (http://arep.med.harvard.edu/ExpressDB/yeastindex.html) of $\sim$ 6500 yeast genes in each. We list the genes that appeared most of the times in top hundred ranks of all these data sets. The genes that appeared

- for five times are YGL130W, YGR246C, and YHR144C and
- for four times are YBR090C, YLR104W, YML021C, YER050C, YER166W, YHR195W, YDR350C, YER155C, YGL067W, YNR052C, YDR285W, YEL042W, YPL106C, and YJL177W.

21.4.4 Average Number of Gene Connections in CoE Networks

From the CoE networks analyzed herein, we estimated the average number of connections per gene as eight and three to seven for human and yeast genome data, respectively (see Fig. 21.3). These findings agree with other published results, where 15 interactions per human protein were identified in the best samples of the

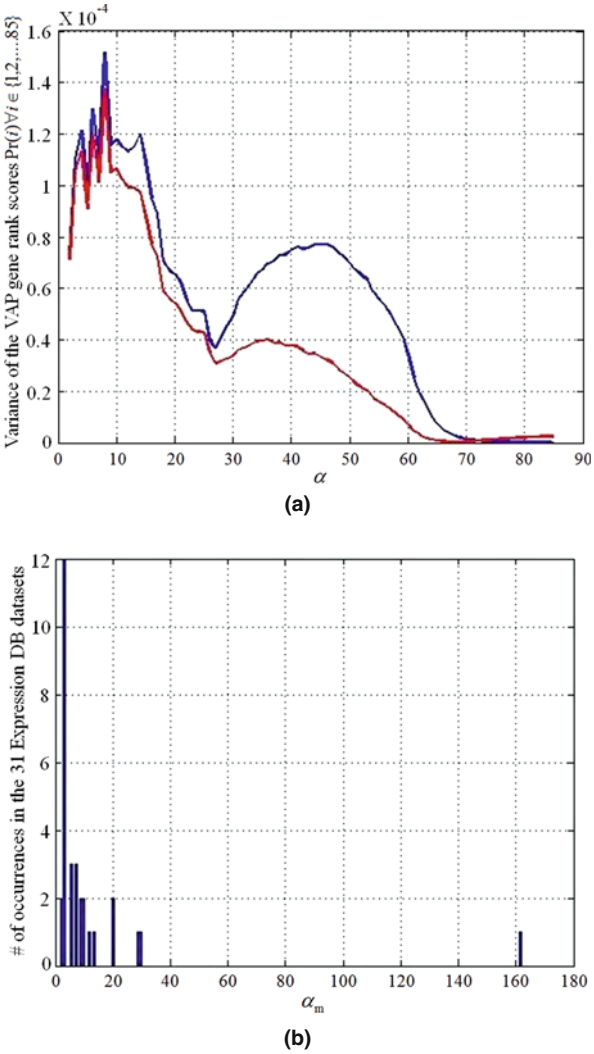


Fig. 21.3 (**a**) Variance of the VAP gene-rank scores $\Pr(i)$ $(\forall\, i \in \{1, 2, \ldots, G\})$ as a function of α where the peak is attained at $\alpha_{\mathrm{m}} = 8$. The *warm grey and cool grey curves* show the results using the matrices $\boldsymbol{W}_{\mathrm{c}}$ and $\boldsymbol{W}_{\mathrm{l}}$, respectively. (**b**) Histogram of the estimated α_{m}s from the 31 ExpressDB data sets

interaction data sets, the Reactome project [6, 8]. Note that we examined only a small fraction of the human genome. Similarly, a recent report found approximately five to ten unique interactions per yeast protein [8]. Thus these reports support the notion of linking the traits of "connectedness" and "importance" as the tendency for human genes to be essentially correlated well with the number of protein–protein partners [8, 9].

21.5 Conclusion

We developed a modified Page-Rank algorithm to rank genes in CoE networks using the fact that every gene is connected to an average number of neighbors in the CoE network. We proposed a new method to compute the metric of average connections per gene in CoE networks. We calculated this metric value as eight for human and three to seven for yeast genome data. We showed through examples that our algorithm is efficient for clustering gene data. The canonical pathways formed by the common five top-ranked VAP genes are well described in infection/pneumonia literature (http://www.ingenuity.com/), adding a level of verification to our propositions.

The proposed analogy between web pages and genes provides new insights into gene networks. In future work, we will investigate these aspects, particularly if the damping factor, d, can be varied to affect the positive and negative feedback loops in immunology. We will also investigate analytically our findings on gene-data clustering, which we obtained in the numerical examples. Namely, we will investigate analytical reasons for the appearance of the local minimum α_l between α_m and α_t in Fig. 21.1b. We will also develop an analytical foundation to relate gene ranking using the Page-Rank algorithm and gene-data clustering, following the concept that we presented in Section 21.3.2. Following this foundation, we will further develop a method to cluster microarray-gene data by extending our analysis in Section 21.3.2, and we will address the effectiveness and efficiency of such clustering.

Acknowledgments The work of W. Zhang was supported by the NSF grants IIS-0535257 and DBI-0743797 and a grant from the Alzheimer's Association. The work of J. P. Cobb was supported by the NIH grants R21GM075023 and R01GM59960.

References

1. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine (http://www-db.stanford.edu/~backrub/google.html), *Proceedings of 7th Intlernational Conference of World Wide Web 7*, pages 107–117, 1998.
2. J. Ruan and W. Zhang. Identification and evaluation of functional modules in gene co-expression networks, *RECOMB Satellite Conferences on Systems Biology and Computational Proteomics,* San Diego, CA, Dec. 2006.
3. J. L. Morrison, R. Breitling, D. J. Higham, and D. R. Gilbert. GeneRank: Using search engine technology for the analysis of microarray experiments, *BMC Bioinformatics,* 6:233, 2005.

4. J. E. McDunn, K. D. Husain, A. D. Polpitiya, A. Burykin. J. Ruan, Q. Li, W. Schierding, N. Lin, D. Dixon, W. Zhang, C. M. Coopersmith, W. M. Dunne, M. Colonna, B. K. Ghosh, and J. P. Cobb. Plasticity of the systematic inflammatory response to acute infection during critical illness: Development of the riboleukogram, *PLoS One,* 3(2):e1564, 2008.
5. A. Thalamuthu, I. Mukhopadhyay, X. Zheng, and G. C. Tseng. Evaluation and comparison of gene clustering methods in microarray analysis, *Bioinformatics,* 22(19):2405–2412, 2006.
6. G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G. R. Gopinath, G. R. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein. Reactome: A knowledgebase of biological pathways, *Nucleic Acids Research,* 33:D428–432, 2005.
7. A. K. Ramani, R. C. Bunescu, R. J. Mooney, and E. M. Marcotte. Consolidating the set of known human protein-protein interactions in preparation for large-scale mapping of the human interactome, *Genome Biologoy,* 6(5):R40, 2005.
8. G. T. Hart, A. K. Ramani, and E. M. Marcotte. How complete are current yeast and human protein-interaction networks? *Genome Biologoy,* 7:I20, 2006.
9. R. Guimerà, S. Mossa, A. Turtschi, and L. A. Amaral. The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles, *Proceedings of the National Academy of Science,* 102(22):7794–7799, 2005.

Index

P.S. Yu, et al. (eds.), *Link Mining: Models, Algorithms, and Applications*,
DOI 10.1007/978-1-4419-6515-8, © Springer Science+Business Media, LLC 2010

H

I

S