Fast Algorithms for Querying and Mining Large Graphs

Hanghang Tong

September 2009
CMU-ML-09-112

Carnegie Mellon

Fast Algorithms for Querying and Mining Large Graphs

Hanghang Tong

September 2009
CMU-ML-09-112

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Christos Faloutsos, CMU, Chair
William Cohen, CMU
Jeff Schneider, CMU
Philip S. Yu, UIC

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Copyright © 2009 Hanghang Tong

This research was sponsored by the Lawrence Livermore National Laboratory (DOE/NNSA) under subcontract numbers B580840, B526511, B573565, and B579447, the National Science Foundation under grant numbers IIS0534205 and DBI0640543, and Lehigh University under subcontract C000027761. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: network analysis, querying, mining, complex user specific pattern, trend analysis, scalability, immunization, anomaly detection, proximity , spectral analysis, low-rank-approximation.

This thesis is dedicated to my parents, Yingen Tong and Xiazhen He.

Abstract

Graphs appear in a wide range of settings and have posed a wealth of fascinating problems. In this thesis, we focus on two types of tasks according to the interaction with users: (1) querying (e.g., *given a social network, how to measure the closeness between two persons? how to track it over time?*) and (2) mining (e.g., *how to identify abnormal behaviors of computer networks? In the case of virus attacks, which nodes are the best to immunize?*).

The task of querying includes three sub-tasks. In the first one, we found that many complex user-specific patterns on large graphs can be answered by means of proximity measurement. In other words, *proximity allows us to query large graphs on the atomic level*. We support our claim by conducting three case studies (connection subgraphs, user feedback, and gateway), all of which (despite their diversity) rely on the proximity measurement as their building block. The proposed algorithms are operational, with careful design and numerous optimizations. For the second sub-task, in order to adapt the querying task to time-evolving graphs, we proposed an efficient algorithm to track proximity on time-evolving graphs, which enables us to do trend analysis on the graph level. The proposed algorithm is up to *176x* faster than competitors and has no quality loss. Finally, in order to handle the scalability issue in the task of querying, we developed a family of fast solutions to compute the proximity in several different scenarios. By carefully leveraging some important properties shared by many real graphs (e.g., the block-wise structure, the linear correlation, the skewness of real bipartite graphs, etc), we can often achieve orders of magnitude of speedup with little or no quality loss.

The task of mining also includes three sub-tasks. In the first one, we proposed an algorithm (NetShield) for immunization under the SIS model. While straight-forward methods are computationally intractable ($O(\binom{n}{k}m)$), the proposed algorithm is *near-optimal, fast* (up to 7 orders of magnitude speedup), and *scalable* ($O(nk^2 + m)$). In the second sub-task, we proposed a family of example-based low-rank matrix approximation methods for anomaly detection. The proposed algorithms are provably equal to or better than the best known methods in both space and time, with the same accuracy. On real data sets, it is up to *112x* faster than the best competitors, for the same accuracy. Finally, we showed that graphs also provide a powerful tool to solve some complex problems. As a case study, we proposed a general framework to mine complex time stamped events (e.g., to find similar time stamps, to find abnormal time stamps and to provide interpretations for our findings, etc) by envisioning the problem as a graph analysis problem. We further proposed MT3 to handle multiple-scale analysis, achieving up to *2 orders of magnitude* speedup, with the same quality.

Acknowledgments

First of all, I want to express my deep gratitude to my advisor Christos Faloutsos, for his advice, encouragement, patience, support and (most importantly) his humor. Working with Christos in the past four years and two months is the most amazing thing happened in my academic life.

I had a fantastic group of collaborators: Duen Horng (Polo) Chau, Tina Eliassi-Rad, Brian Gallagher, Jason I. Hong, Hani Jamjoom, Yehuda Koren, Jure Leskovec, Kensuke Onuma, Jia-Yu (Tim) Pan, Spiros Papadimitriou, B. Aditya Prakash, Huiming Qu, José (Junio) Fernando Rodrigues Jr., Yasushi Sakurai, Jimeng Sun, Agma J. M. Traina, Charalampos (Babis) Tsourakakis, and Philip S. Yu. It were Tim and Jimeng who introduced me to the DB group at the first time. Tina always came up with very interesting real problems, with which I never need to worry about '*what-to-work next*'. I learned a lot from Spiros who is always elegant and precise on research. Their insights, feedback, and comments have helped a lot with my thesis!

I had two great experiences for internship during the Ph.D study. I want to thank my mentors and managers there: Yeduda Koren, Stephen C. North and Chris Volinsky from AT&T Shannon Labs; and Huiming Qu and Hani Jamjoom from IBM Watsons Labs. I also want to thank Mingjing Li and Hong-Jiang Zhang from Microsoft Asia, where my academic life starts. Mingjing taught me how to do research hand-by-hand. Many thanks also go to Harry Shum for his great advice before I came to CMU, which was proved to be extremely useful for me.

I want to thank the thesis committee members, William Cohen, Jeff Schneider, and Philip S. Yu for their great comments, insights, and feedback.

I want to thank all the members in the DB group, and the whole machine learning department. I want to specially thank Diane Stidle and Charlotte Yano, who have helped to smooth the life in CMU. Diane always creates a 'home-feeling' atmosphere in the whole department.

Finally, I want to thank my wife (Jingrui He) and my three-month-old daughter (Emma Tong), who are the ultimate origin where all my motivation, confidence, strength and inspiration come from.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Graphs appear in a wide range of settings and account for a large portion of real word data sets [Cha05]. For example, in sociology, the nodes are individuals and the edges represent the interaction between two persons (e.g., collaboration, trust, contact, etc); in computer networks, the nodes are routers or autonomous systems and edges represent the connection between two routers/automonous systems; in user psychology, the nodes are people and items, and the edges represent some actions between the user and the items (e.g., the user *clicks* the web page, the user *recommends* some product, etc.); in ecology, the nodes are species, and edges represent prey-predator relationship; in biology, the nodes are proteins and the edges represent the interaction between two proteins (e.g., both are critical for some biological process to happen).

Such graphs have posed a wealth of fascinating research questions. To name a few, given a social network, how to measure the closeness (i.e., proximity, relevance, etc) between two persons, and how to track it over time? Given a customer-question in a help center, who is the best expert to route it to? How to identify abnormal behaviors of computer networks? In the case of virus attacks, which nodes are the best to immunize? etc.

Answering these questions are critical for many real high impact applications. For example, proximity measurement and tracking are crucial for querying/exploring large graphs, which play an important role in on-line social networks; anomaly detection in terrorist networks as well as computer networks is vital for national security; immunization is crucial to defend networks in the case of a virus attack; a good immunization strategy might be also very helpful for designing a good k-advertisement strategy in viral marketing.

In this thesis, we address the above challenges in multiple dimensions, by focusing on two types of tasks according to the interaction with users: querying and mining. For the task of querying, we want to answer the complex user-specific patterns, such as Center-Piece Subgraphs (*Given three criminals, who is the master-mind?*). We also want to track proximity on time-evolving graphs (*How close is author 'Smith' to the 'KDD' conference, and how is this changing over time?*). For the task of mining, the goal is to summarize/compress a graph, and report anomalies. For each task, we further address three sub-tasks, which are summarized in table 1.1.

Table 1.1: Thesis Overview: Tasks

Querying	**Q1**: Finding complex user-specific patterns, e.g., • **Q1.1.** Center-Piece Subgraphs Discovery (Chapter 3) • **Q1.2.** Querying with User Feedback (Chapter 4) • **Q1.3.** Gateway Detection (Chapter 5) **Q2**: Querying over time (Chapter 6) **Q3**: Scalability (Chapters 2-6, 9)
Mining	**M1**: Vulnerability Analysis (Chapter 7) **M2**: Anomaly Detection (Chapter 8) **M3**: Mining Complex Time Stamped Events (Chapter 9)

1.2 Impact, Applications and Main Contributions

We make the following key contributions in the thesis. The more detailed contributions, mapping to the specific applications are summarized in table 1.2.

1 Since node proximity is at the heart of several of the above problems, we carefully designed node proximity algorithms, which are both fast and effective.

2 We proposed fast algorithms to numerous, real-life graph problems (center-piece subgraphs, querying with user feedback, gateway finder, proximity tracking, immunization, and low-rank approximation, etc).

3 We provided numerous proofs to illustrate the *correctness* (e.g., Theorem 4, Theorem 6, Theorem 10, etc), *accuracy* (e.g., Theorem 1, Theorem 5, Lemma 4, etc) and *computational complexity* in big-O notations (e.g., Lemma 7, Lemma 12, Lemma 16, Lemma 17, etc) of our algorithms

4 We conducted experiments on numerous real data sets, most of which are publicly available, illustrating the speed and accuracy of our algorithms.

1.3 Thesis Organization

Table 1.3 gives an overview of the thesis work. In the following chapters, we will describe our work in details. We will start with proximity definitions and fast solutions in Part I, which is the main tool for querying large graphs. Then in Part II, we present our work on querying static graphs by three case studies. In Part III, we address the problem of how to query dynamic graphs. We present our work on mining graphs in Part IV and Part V. Finally, we conclude the thesis in chapter 10.

Table 1.2: Impact, Applications and Main Contributions of Thesis Work

Tasks		Impact and Applications	Our Main Contributions
Q1	Q 1.1	•Find common advisor •Find master-mind criminal •Find similar gene	•Problem definitions for Center-Piece Subgraphs (CePS) •An algorithm to find CePS *(effective and fast)*
	Q 1.2	•Interactive neighbor search •Interactive CePS •Interactive recommendation •Semi-supervised image captions	• A novel method (iPoG) to incorporate user feedback in measuring node proximity • A fast algorithm to compute iPoG *(fast, little quality loss)*
	Q 1.3	•Skill search •Question re-route	•A novel 'gateway-ness' score •Two algorithms to find a set of nodes with the highest 'gateway-ness' score *(near-optimal, fast and scalable)*
Q2		•Scale sophisticated trend analysis to time-evolving graphs	•Novel proximity and centrality definitions for time-evolving graphs •Two fast update algorithms *(no quality loss and fast)*
Q3		•Enable Q1 and Q2 to large graphs	A family of fast solutions to compute RWR *(orders of magnitude speedup, little or no quality loss)*
S1		• Immunization under SIS model • Summarize graphs by sketch	•A Novel definition to measure the 'bridgeness' on graphs •An algorithm to find a set of nodes with the highest 'bridgeness' score *(near-optimal, fast and scalable)*
S2		•Detect anomalies on graphs	A family of example-based low-rank approximation methods *(equal accuracy; equal or better speed and space; easy to update; intuitive to interpret)*
S3		•Find time cluster, •Find abnormal time stamp •Provide interpretation	•A generic framework to mine complex time stamped events •An efficient algorithm for multiple scale analysis *(fast, no quality loss)*

Table 1.3: Thesis Overview: Organization

Tasks	Types of Graphs: Static	Types of Graphs: Dynamic
Querying	Q1 Chapter 2-5	Q2 Chapter 2,6
	Q3 Chapter 2-5	Q3 Chapter 2,6
Mining	M1 Chapter 7	M3 Chapter 2,9
	M2 Chapter 8	M2 Chapter 8

Part I

Fundamentals: Proximity Definitions and Fast Solutions

Chapter 2

Proximity Definitions and Fast Solutions

Summary of This Chapter

- **Questions we want to answer:**

  Q1: How to quantify the closeness/ relevance between two nodes (or two groups of nodes in the graph)?

  Q2: How to compute it fast?

- **Our answers and contributions**

  A1: We suggest using random walk with restart as the basic solution, and then propose directionality-aware proximity and their generalizations.

  A2: We proposed a family of fast solutions, which achieves orders of magnitude speed up, with little or no quality loss.

2.1 Introduction

Measuring the proximity (i.e., relevance/closeness) score between two nodes is one of the fundamental building blocks for querying and mining graphs [ABC+02] [LNK03] [LJM+07] [FMT04] [PRTU05] [SQCF05]. It is the main tool behind all the querying tasks and some mining tasks of this thesis. For example, figure (2.1-a) shows some results for the auto-captioning application as in [PYFD04]. We will present more case studies in Part II.

In this chapter, we mainly focus on the following two questions:

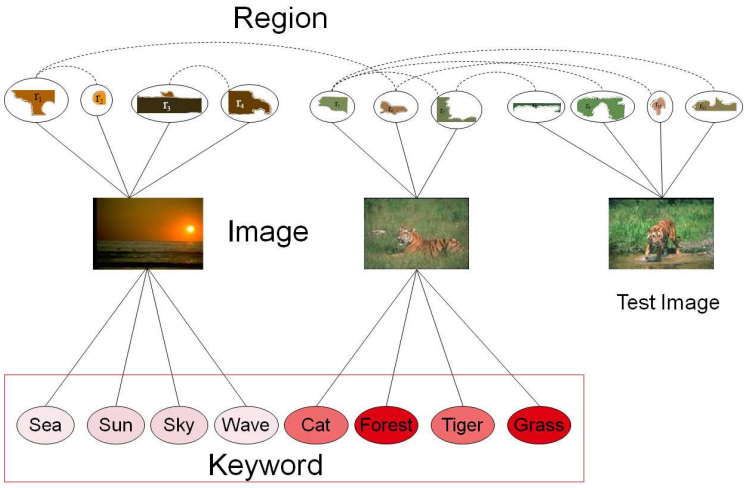
Q1 How to define a good proximity measurement?

Q2 How to compute it fast in large graphs?

For many real graphs, the relationship between two nodes (e.g., the relationship between two persons on a social network) often exhibits multiple-facets. Traditional graph distance measurements (e.g., shortest path, maximum flow) fail to capture such characteristic. To address this issue, we suggest use random walk with restart (RWR) as a basic proximity measurement, which is able to summarize the multiple weighted connections between two nodes on the graphs.

‘Jet’ ‘Plane’ ‘Runway’ ‘Texture’ ‘Candy’ ‘Background’

(a) Some captioned images by gCap [PYFD04].



(b) Underlying graph used for image caption. See details in [PYFD04].

Figure 2.1: Using proximity measurement for image caption.

In terms of computational cost, RWR requires a matrix inversion. There are two straightforward solutions, none of which is scalable for large graphs: The first one is to pre-compute and store the inversion of a matrix (“PreCompute” method); the second one is to compute the matrix inversion on the fly, say, through power iteration (“OnTheFly” method). The first method is fast at query time, but prohibitive in terms of space (quadratic on the number of nodes on the graph), while the second is slow at query time.

Here we propose a novel solution to this challenge. Our approach, B_LIN, takes the advantage of two properties shared by many real graphs: (a) the block-wise, community-like structure, and (b) the linear correlations across rows and columns of the adjacency matrix. The proposed method carefully balances the off-line pre-processing cost (both the CPU cost and the storage cost), with

the response quality (with respect to both the accuracy and the response time). Compared to PreCompute, it only requires pre-computing and storing the low-rank approximation of a large but sparse matrix, and the inversion of some small size matrices. Compared with OnTheFly, it only need a few matrix-vector multiplication operations in on-line response process.

The main contributions of this chapter are as follows:

- A novel, fast, and practical solution (B_LIN and its derivatives, NB_LIN and BB_LIN);
- Theoretical justification and analysis, giving an error bound for NB_LIN;
- Extensive experiments on several typical applications, with real data.

The proposed method is operational, with careful design and numerous optimizations. Our experimental results show that, in general, it preserves 90%+ quality, while (a) saves several orders of magnitude of pre-computation and storage cost over PreCompute, and (b) it achieves up to 150x speedup on query time over OnTheFly. For the DBLP author-conference dataset, with light pre-computational and storage cost, it achieves up to 1,800x speedup with *no quality loss*.

The rest of this chapter is organized as follows: we review random walk with restart and analyze its computational challenges in Section 2.2; the proposed method is presented in Section 2.3; the justification and the analysis are provided in Section 2.4. The experimental results are presented in Section 2.5. The related work is given in Section 2.6. Finally, we conclude the paper in Section 2.7.

2.2 Preliminaries

Table 2.1 gives a list of symbols used in this chapter. In this Section, we first introduce random walk with restart and explain why it is a good proximity measurement.

2.2.1 Preliminary # 1: Random Walk with Restart

One of the most popular way to measure the proximity is random walk with restart, which is defined as equation (2.1) [PYFD04]: consider a random particle that starts from node i. The particle iteratively transmits to its neighborhood with the probability that is proportional to their edge weights. Also at each step, it has some probability c to return to the node i. The relevance score of node j wrt node i is defined as the steady-state probability $r_{i,j}$ that the particle will finally stay at node j [PYFD04].

$$\vec{r_i} = c\tilde{\mathbf{W}}\vec{r_i} + (1-c)\vec{e_i} \tag{2.1}$$

Equation (2.1) defines a linear system problem, where $\vec{r_i}$ is determined by:

$$\begin{aligned} \vec{r_i} &= (1-c)(I - c\tilde{\mathbf{W}})^{-1}\vec{e_i} \\ &= (1-c)\mathbf{Q}^{-1}\vec{e_i} \end{aligned} \tag{2.2}$$

The relevance score defined by RWR has many good properties: compared with those pair-wise metrics, it can capture the global structure of the graph [HLZ+04]; compared with those

Table 2.1: Symbols

Symbol	Definition
$\mathbf{W} = [w_{i,j}]$	the weighted graph, $1 \leq i, j \leq n$
$\tilde{\mathbf{W}}$	the normalized weighted matrix associated with $\mathbf{W}$
$\tilde{\mathbf{W}}_1$	the within-partition matrix associated with $\tilde{\mathbf{W}}$
$\tilde{\mathbf{W}}_2$	the cross-partition matrix associated with $\tilde{\mathbf{W}}$
$\mathbf{Q}$	the system matrix associated with $\mathbf{W}$: $\mathbf{Q} = \mathbf{I} - c\tilde{\mathbf{W}}$
$\mathbf{D}$	$n \times n$ matrix, $D_{i,i} = \sum_j w_{i,j}$ and $D_{i,j} = 0$ for $i \neq j$
$\mathbf{U}$	$n \times t$ node-concept matrix
$\mathbf{S}$	$t \times t$ concept-concept matrix
$\mathbf{V}$	$t \times n$ concept-node matrix
$\mathbf{0}$	a block matrix, whose elements are all zeros
$\vec{e}_i$	$n \times 1$ starting vector, the i^{th} element 1 and 0 for others
$\vec{r}_i = [r_{i,j}]$	$n \times 1$ ranking vector, $r_{i,j}$ is the relevance score of node j wrt node i
c	the restart probability, $0 \leq c \leq 1$
n	the total number of the nodes in the graph
k	the number of partitions
t	the rank of low-rank approximation
m	the maximum iteration number
ξ_1	the threshold to stop the iteration process
ξ_2	the threshold to sparse the matrix

traditional graph distances (such as shortest path, maximum flow etc), it can capture the multi-facet relationship between two nodes [TF06].

2.2.2 Preliminary # 2: Computational Challenges

One of the most widely used ways to solve random walk with restart is the iterative method, iterating the equation (2.1) until convergence, that is, until the L_2 norm of successive estimates of $\vec{r}_i$ is below our threshold ξ_1, or a maximum iteration step m is reached. In the chapter, we refer to it as OnTheFly method. OnTheFly does not require pre-computation and additional storage cost. Its on-line response time is linear to the iteration number and the number of edges[1], which might be undesirable when (near) real-time response is a crucial factor while the data set is large. A nice observation of [SQCF05] is that the distribution of $\vec{r}_i$ is highly skewed. Based on this observation, combined with the factor that many real graphs has block-wise/community structure, the authors in [SQCF05] proposed performing RWR only on the partition that contains the starting point i (method *Blk*). However, for all data points outside the partition, $r_{i,j}$ is simply set 0. In other words, *Blk* outputs a local estimation of $\vec{r}_i$.

[1]Here, we store $\tilde{\mathbf{W}}$ in a sparse format.

Table 2.2: B_LIN

Input: The normalized weighted matrix $\tilde{\mathbf{W}}$ and the starting vector $\vec{e}_i$
Output: The ranking vector $\vec{r}_i$
Pre-Computational Stage(Off-Line):
p1. Partition the graph into k partitions by METIS [KK99];
p2. Decompose $\tilde{\mathbf{W}}$ into two matrices: $\tilde{\mathbf{W}} = \tilde{\mathbf{W}}_1 + \tilde{\mathbf{W}}_2$ according to the partition result, where $\tilde{\mathbf{W}}_1$ contains all within-partition links and $\tilde{\mathbf{W}}_2$ contains all cross-partition links;
p3. Let $\tilde{\mathbf{W}}_{1,i}$ be the i^{th} partition, denote $\tilde{\mathbf{W}}_1$ as equation(2.3);
p4. Compute and store $\mathbf{Q}_{1,i}^{-1} = (\mathbf{I} - c\tilde{\mathbf{W}}_{1,i})^{-1}$ for each partition i;
p5. Do low-rank approximation for $\tilde{\mathbf{W}}_2 = \mathbf{USV}$;
p6. Define $\mathbf{Q}_1^{-1}$ as equation (2.4). Compute and store $\tilde{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - c\mathbf{V}\mathbf{Q}_1^{-1}\mathbf{U})^{-1}$.
Query Stage (On-Line):
q1. Output $\vec{r_i} = (1-c)(\mathbf{Q}_1^{-1}\vec{e_i} + c\mathbf{Q}_1^{-1}\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\mathbf{Q}_1^{-1}\vec{e_i})$.

On the other hand, it can be seen from equation (2.2) that the system matrix $\mathbf{Q}$ defines all the steady-state probabilities of random walk with restart. Thus, if we can pre-compute and store $\mathbf{Q}^{-1}$, we can get $\vec{r}_i$ real-time (We refer to this method as PreCompute). However, pre-computing and storing $\mathbf{Q}^{-1}$ is impractical when the dataset is large, since it requires quadratic space and cubic pre-computation. [2]

On the other hand, linear correlations exist in many real graphs, which means that we can approximate $\tilde{\mathbf{W}}$ by low-rank approximation. This property allows us to approximate $\mathbf{Q}^{-1}$ very efficiently. Moreover, this enables a global estimation of $\vec{r}_i$, unlike the local estimation obtained by *Blk*. However, due to the low rank approximation, such kind of estimation is conducted at a coarse resolution.

2.3 Proposed Fast Solutions

2.3.1 Proposed Algorithm

In summary, the skewed distribution of $\vec{r}_i$ and the block-wise structure of the graph lead to a local/fine resolution estimation; the linear correlations of the graph lead to a global/coarse resolution estimation. In this chapter, we combine these two properties in a unified manner. The proposed algorithm, B_LIN is shown in table 2.2. A pictorical description of B_LIN is given in figure 2.2.

[2] Even if we use OnTheFly to compute each column of $\mathbf{Q}^{-1}$, the pre-computation cost is $O(nm)$.

$$\tilde{\mathbf{W}}_1 = \begin{pmatrix} \tilde{\mathbf{W}}_{1,1} & \mathbf{0} & ... & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{W}}_{1,2} & ... & \mathbf{0} \\ ... & ... & ... & ... \\ \mathbf{0} & ... & \mathbf{0} & \tilde{\mathbf{W}}_{1,k} \end{pmatrix} \tag{2.3}$$

$$\mathbf{Q}_1^{-1} = \begin{pmatrix} \mathbf{Q}_{1,1}^{-1} & \mathbf{0} & ... & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{1,2}^{-1} & ... & \mathbf{0} \\ ... & ... & ... & ... \\ \mathbf{0} & ... & \mathbf{0} & \mathbf{Q}_{1,k}^{-1} \end{pmatrix} \tag{2.4}$$

2.3.2 Normalization on $\mathbf{W}$

B_LIN takes the normalized matrix $\tilde{\mathbf{W}}$ as the input. There are several ways to normalize the weighted matrix $\mathbf{W}$. The most natural way might be by row normalization [PYFD04]. Complementarily, the authors in [ZBL+03] propose using the normalized graph Lapalician ($\tilde{\mathbf{W}} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$). In [TF06], the authors also propose penalizing the famous nodes before row normalization for social network.

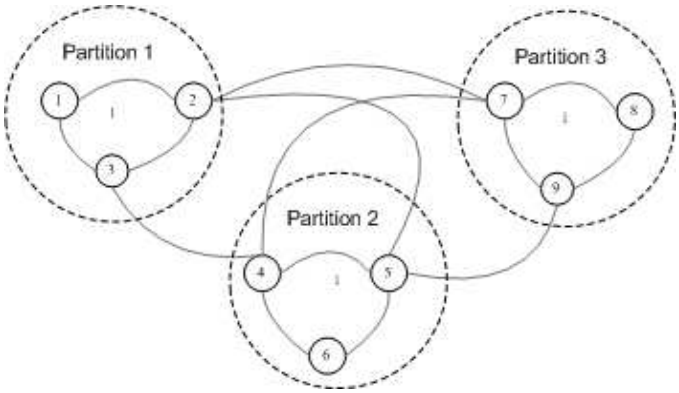
It should be pointed out that all the above normalization methods can be fitted into the proposed B_LIN. However, in this chapter, we will focus on the normalized graph Laplacian[3] for the following reasons:

- For real applications, these normalization methods often lead to very similar results. (For cross-media correlation discovery, our experiments demonstrate that normalized graph Laplacian actually outperforms the row normalization method, which is originally proposed by the authors in [PYFD04]
- Unlike the other two methods, normalized graph Laplacian outputs the symmetric relevance score (that is $r_{i,j} = r_{j,i}$), which is a desirable property for some applications.
- The normalized graph Laplacian is symmetric, and it leads to a symmetric $\mathbf{Q}_1$, which will save 50% storage cost.
- It might be difficult to develop an error bound for B_LIN in the general case. However, as we will show in Section 3.3, it is possible to develop an error bound for the simplified version (NB_LIN) of B_LIN, which also benefits from the symmetric property of the normalized graph Laplacian.

2.3.3 Discussion of Partition number k

The partition number k balances the complexity of $\tilde{\mathbf{W}}_1$ and $\tilde{\mathbf{W}}_2$. We will evaluate different values for k in the experiment section. Here, we investigate two extreme cases of k.

[3]It should be pointed out that strictly speaking, $\vec{r}_i$ is no longer a probability distribution. However, for all the applications we cover in this chapter, it does not matter since what we need is a relevance score. On the other hand, we can always normalized $\vec{r}_i$ to get a probability distribution.



(a) Original weighted graph, consisting of 3 partitions, which are indicated by the dash circles.

$$\begin{pmatrix} \\ \bar{W} \\ \\ \end{pmatrix} = \begin{pmatrix} \bar{W}_{1,1} & 0 & 0 \\ 0 & \bar{W}_{1,2} & 0 \\ 0 & 0 & \bar{W}_{1,3} \end{pmatrix} + \begin{pmatrix} \\ U \\ \\ \end{pmatrix} (S) (\quad V \quad)$$

(b) Decompose original weighted graph into within-partition matrix ($\tilde{\mathbf{W}}_1$), which is block-diagonal, and cross-partition matrix, which is approximated by low-rank approximation ($\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$).

$$\begin{pmatrix} \\ I - c\bar{W} \\ \\ \end{pmatrix}^{-1} = \begin{Bmatrix} \begin{pmatrix} Q_{1,1}^{-1} & 0 & 0 \\ 0 & Q_{1,2}^{-1} & 0 \\ 0 & 0 & Q_{1,3}^{-1} \end{pmatrix} \\ + \\ c \begin{pmatrix} Q_{1,1}^{-1} & 0 & 0 \\ 0 & Q_{1,2}^{-1} & 0 \\ 0 & 0 & Q_{1,3}^{-1} \end{pmatrix} \begin{pmatrix} \\ U \\ \\ \end{pmatrix} (\bar{\Lambda}) (\quad V \quad) \begin{pmatrix} Q_{1,1}^{-1} & 0 & 0 \\ 0 & Q_{1,2}^{-1} & 0 \\ 0 & 0 & Q_{1,3}^{-1} \end{pmatrix} \end{Bmatrix}$$

(c) Approximate the inverse of $(\mathbf{I} - c\tilde{\mathbf{W}})$ by the inversion of a few small size matrices($\mathbf{Q}_{1,1}$, $\mathbf{Q}_{1,2}$, $\mathbf{Q}_{1,3}$ and $\tilde{\mathbf{\Lambda}}$),which can be pre-computed and stored more efficiently.

Figure 2.2: A pictorical description of B_LIN

Table 2.3: NB_LIN

Input: The normalized weighted matrix $\tilde{\mathbf{W}}$ and the starting vector $\vec{e}_i$
Output: The ranking vector $\vec{r}_i$
Pre-Computational Stage(Off-Line):
p1. Do low-rank approximation for $\tilde{\mathbf{W}} = \mathbf{USV}$;
p2. Compute and store $\tilde{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - c\mathbf{VU})^{-1}$.
Query Stage (On-Line):
q1. Output $\vec{r}_i = (1-c)(\vec{e}_i + c\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\vec{e}_i)$.

Table 2.4: BB_LIN

Input: The normalized weighted matrix $\tilde{\mathbf{W}}$ and the starting vector $\vec{e}_i$ as equation(2.5)
Output: The ranking vector $\vec{r}_i$ as equation(2.5)
Pre-Computational Stage(Off-Line):
p1. Compute and store $\tilde{\mathbf{\Lambda}} = (\mathbf{I} - c^2\mathbf{M}^T\mathbf{M})^{-1}$;
Query Stage (On-Line):
q1. $\vec{r}_{i,1} = (1-c)(\vec{e}_{i,1} + c^2\mathbf{M}\tilde{\mathbf{\Lambda}}\mathbf{M}^T\vec{e}_{i,1} + c\mathbf{M}\tilde{\mathbf{\Lambda}}\vec{e}_{i,2})$
q2. $\vec{r}_{i,2} = (1-c)(c\tilde{\mathbf{\Lambda}}\mathbf{M}^T\vec{e}_{i,1} + \tilde{\mathbf{\Lambda}}\vec{e}_{i,2})$
q3. Output $\vec{r}_i = (\vec{r}_{i,1}, \vec{r}_{i,2})^T$.

First, if $k = 1$, we have $\tilde{\mathbf{W}}_1 = \tilde{\mathbf{W}}$ and $\tilde{\mathbf{W}}_2 = \mathbf{0}$. Then, B_LIN is just equivalent to the PreCompute method.

On the other hand, if $k = n$, we have $\tilde{\mathbf{W}}_1 = \mathbf{0}$ and $\tilde{\mathbf{W}}_2 = \tilde{\mathbf{W}}$. In this case, $\mathbf{Q}_1 = \mathbf{I}$ and we have the following simplified version of B_LIN as in table 2.3. We refer it as NB_LIN.

An application of random walk with restart is neighborhood formulation in the bipartite graph [SQCF05]. Suppose there are n_1 and n_2 nodes for each type of objects in the bipartite graph; $\mathbf{M}$ is the $n_1 \times n_2$ bipartite matrix. The normalized weighted matrix, the starting vector and the ranking vector have the following format:

$$\tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{0} & \mathbf{M} \\ \mathbf{M^T} & \mathbf{0} \end{pmatrix} \quad \vec{r}_i = \begin{pmatrix} \vec{r}_{i,1} \\ \vec{r}_{i,2} \end{pmatrix} \quad \vec{e}_i = \begin{pmatrix} \vec{e}_{i,1} \\ \vec{e}_{i,2} \end{pmatrix} \tag{2.5}$$

As a direct application of NB_LIN, we have the following fast algorithm (BB_LIN) for one class of bipartite graph when $n_1 \gg n_2$ as in table (2.4)

2.3.4 Low-rank approximation on $\tilde{\mathbf{W}}_2$

One natural choice to do low-rank approximation on $\tilde{\mathbf{W}}_2$ is by eigen-value decomposition[4]:

$$\tilde{\mathbf{W}}_2 = \mathbf{USU}^T \tag{2.6}$$

[4]if the other two normalization methods are used, we can do singular vector decomposition instead.

Table 2.5: Low Rank Approximation by Partition

Input: The cross-partition matrix $\tilde{\mathbf{W}}_2$ and t
Output: Low rank approximation of $\tilde{\mathbf{W}}_2$: $\mathbf{U}, \mathbf{S}, \mathbf{V}$
1. Partition $\tilde{\mathbf{W}}_2$ into t partitions;
2. Construct an $n \times t$ matrix $\mathbf{U}$. The i^{th} column of $\mathbf{U}$ is the sum of all the columns of $\tilde{\mathbf{W}}_2$ that belong to the i^{th} partition;
3. Compute $\mathbf{S} = (\mathbf{U}^T\mathbf{U})^{-1}$;
4. Compute $\mathbf{V} = \mathbf{U}^T\tilde{\mathbf{W}}_2$.

where each column of $\mathbf{U}$ is the eigen-vector of $\tilde{\mathbf{W}}_2$ and $\mathbf{S}$ is a diagonal matrix, whose diagonal elements are eigen-values of $\tilde{\mathbf{W}}_2$.

The advantage of eigen-value decomposition is that it is 'optimal' in terms of reconstruction error. Also, since $\mathbf{V} = \mathbf{U}^T$ in this situation, we can save 50% storage cost. However, one potential problem is that it might lose the sparsity of original matrix $\tilde{\mathbf{W}}_2$. Also, when $\tilde{\mathbf{W}}_2$ is large, doing eigen-value decomposition itself might be time-consuming.

To address this issue, in this chapter, we also propose the following heuristic to do low-rank approximation as in table 2.5. Its basic idea is that, firstly, construct $\mathbf{U}$ by partitioning $\tilde{\mathbf{W}}_2$; and then use the projection of $\tilde{\mathbf{W}}_2$ on the sub-space spanned by the columns of $\mathbf{U}$ as the low-rank approximation.

2.4 Justification and Analysis

2.4.1 Correctness

Here, we present a brief proof of the proposed algorithms.

Correctness of B_LIN

Lemma 1. *If $\tilde{\mathbf{W}} = \tilde{\mathbf{W}}_1 + \mathbf{USV}$ holds, B_LIN outputs exactly the same result as PreCompute.*
Proof: Since $\tilde{\mathbf{W}}_1$ is a block-diagonal matrix. Based on equation (2.3) and (2.4), we have

$$(\mathbf{I} - c\tilde{\mathbf{W}}_1)^{-1} = \mathbf{Q}_1^{-1} \tag{2.7}$$

Then, based on the Sherman-Morrison lemma [PC90], we have:

$$\begin{aligned} \tilde{\mathbf{\Lambda}} &= (\mathbf{S}^{-1} - c\mathbf{V}\mathbf{Q}_1^{-1}\mathbf{U})^{-1} \\ (\mathbf{I} - c\tilde{\mathbf{W}})^{-1} &= (\mathbf{I} - c\tilde{\mathbf{W}}_1 - c\mathbf{USV})^{-1} \\ &= \mathbf{Q}_1^{-1} + c\mathbf{Q}_1^{-1}\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\mathbf{Q}_1^{-1} \\ \vec{r_i} &= (1-c)(\mathbf{Q}_1^{-1}\vec{e_i} + c\mathbf{Q}_1^{-1}\mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{V}\mathbf{Q}_1^{-1}\vec{e_i}) \end{aligned}$$

which completes the proof of Lemma 1 □ .

It can be seen that the only approximation of B_LIN comes from the low-rank approximation for $\tilde{\mathbf{W}}_2$.

We can also interpret B_LIN from the perspective of latent semantic/concept space. By low-rank approximation on $\tilde{\mathbf{W}}_2$, we actually introduce a $t \times t$ latent concept space by $\mathbf{S}$. Furthermore, if we treat the original $\tilde{\mathbf{W}}$ as an $n \times n$ node space, $\mathbf{U}$ and $\mathbf{V}$ actually define the relationship between these two spaces ($\mathbf{U}$ for node-concept relationship and $\mathbf{V}$ for concept-node relationship). Thus, it can be seen that, instead of doing random walk with restart on the original whole node space, B_LIN decomposes it into the following simple steps:

(1) Doing RWR within the partition that contains the starting point (multiply $\vec{e_i}$ by $\mathbf{Q}_1^{-1}$);

(2) Jumping from node-space to latent concept space (multiply the result of (1) by $\mathbf{V}$);

(3) Doing RWR within the latent concept space (multiply the result of (2) by $\tilde{\mathbf{\Lambda}}$);

(4) Jumping back to the node space(multiply the result of (3) by $\mathbf{U}$);

(5) Doing RWR within each partition until convergence (multiply the result of (4) by $\mathbf{Q}_1^{-1}$).

Correctness of NB_LIN

Lemma 2. *If $\tilde{\mathbf{W}} = \mathbf{USV}$ holds, NB_LIN outputs exactly the same result as PreCompute.*

Proof: Taking $\tilde{\mathbf{W}}_1 = \mathbf{0}$ and $\mathbf{Q}_1 = \mathbf{I}$, by applying Lemma 1, we directly complete the proof of Lemma 2. which completes the proof. □

Correctness of BB_LIN

Lemma 3. *BB_LIN outputs exactly the same result as PreCompute.*

Proof: Substituting equation (2.5) into equation (2.2), we have

$$\begin{aligned} \vec{r}_{i,1} &= (1-c)(\mathbf{I} - c^2\mathbf{M}\mathbf{M}^T)^{-1}(c\mathbf{M}\vec{e}_{i,2} + \vec{e}_{i,1}) \\ \vec{r}_{i,2} &= (1-c)(\mathbf{I} - c^2\mathbf{M}^T\mathbf{M})^{-1}(c\mathbf{M}^T\vec{e}_{i,1} + \vec{e}_{i,2}) \end{aligned}$$

Solving $\vec{r}_{i,2}$ directly completes the proof of 'q2' in table (2.4).

Define a new RWR, which takes 1) $(c\mathbf{M}\vec{e}_{i,2}+\vec{e}_{i,1})$ as the new starting vector; 2) $(c\mathbf{M}\mathbf{M}^T)$ as the new normalized weighted matrix; and 3) $(\mathbf{M}(c\mathbf{I})\mathbf{M}^T)$ as the low-rank approximation. Applying Lemma 2 to this RWR, we complete the proof for 'q1' in table (2.4), which in turn completes the proof of Lemma 3. □

2.4.2 Computational and storage cost

In this section, we make a brief analysis for the proposed algorithms in terms of computational and storage cost. For the limited space, we only provide the result for B_LIN.

On-line computational cost

It is not hard to see that, at the on-line query stage of B_LIN (table 2.2, step q1), we only need a few matrix-vector multiplication operations as shown in equation (2.8). Therefore, B_LIN is capable of meeting the fast response requirement.

$$\begin{aligned} \vec{r}_0 &\leftarrow \mathbf{Q}_1^{-1}\vec{e}_i \\ \vec{r}_i &\leftarrow \mathbf{V}\vec{r}_0 \\ \vec{r}_i &\leftarrow \tilde{\mathbf{\Lambda}}\vec{r}_i \\ \vec{r}_i &\leftarrow \mathbf{U}\vec{r}_i \\ \vec{r}_i &\leftarrow \mathbf{Q}_1^{-1}\vec{r}_i \\ \vec{r}_i &\leftarrow (1-c)(\vec{r}_0 + c\vec{r}_i) \end{aligned} \tag{2.8}$$

Pre-computational cost

The main off-line computational cost of the proposed algorithm consists of the following parts:

(1) partitioning the whole graph;

(2) inversion of each $\mathbf{I} - c\tilde{\mathbf{W}}_{1,i}, (i = 1, ..., k)$;

(3) low-rank approximation on $\tilde{\mathbf{W}}_2$;

(4) inversion of $(\mathbf{S}^{-1} - \mathbf{V}\mathbf{Q}_1^{-1}\mathbf{U})$.

Thus, instead of solving the inversion of the original $n \times n$ matrix, B_LIN (1) inverts $k + 1$ small matrices ($\mathbf{Q}_{1,i}^{-1}$, i=1,...,k, and $\tilde{\mathbf{\Lambda}}$); (2) computes a low-rank approximation of a sparse $n \times n$ matrix ($\tilde{\mathbf{W}}_2$), and (3) partitions the whole graph.

Pre-storage cost

In terms of storage cost, we have to store $k + 1$ small matrices ($\mathbf{Q}_{1,i}^{-1}, (i = 1, ..., k)$, and $\tilde{\mathbf{\Lambda}}$), one $n \times t$ matrix ($\mathbf{U}$) and one $t \times n$ matrix ($\mathbf{V}$). Moreover, we can further save the storage cost as shown in the following:

- An observation from all our experiments is that many elements in $\mathbf{Q}_{1,i}^{-1}$, $\mathbf{U}$ and $\mathbf{V}$ are near zeros. Thus, an optional step is to set these elements to be zero (by the threshold ξ_2) and to store these matrices as sparse format. For all experiments in this chapter, we find that this step will significantly reduce the storage cost while almost not affecting the approximation accuracy.
- The normalized graph Laplacian is symmetric, which leads to (1) a symmetric $\mathbf{Q}_{1,i}^{-1}$, and (2) $\mathbf{U} = \mathbf{V}^T$, if eigen-value decomposition is used when computing the low-rank approximation[5]. By taking advantage of this symmetry property, we can further save 50% storage cost.

[5]On the other hand, if we use partition-based low-rank approximation as in table (2.5), $\mathbf{U}$ and $\mathbf{V}$ are usually sparse and thus can be efficiently stored

2.4.3 Error Bound for NB_LIN

Developing an error bound for the general case of the proposed methods is difficult. However, for NB_LIN (table 2.3), we have the following lemma:

Lemma 4. *Let $\vec{r}$ and $\hat{\vec{r}}$ be the ranking vectors* [6] *by PreCompute and by NB_LIN, respectively. If NB_LIN takes eigen-value decomposition as low-rank approximation, $\|\vec{r}-\hat{\vec{r}}\|_2 \leq (1-c)\sum_{i=t+1}^{n}\frac{1}{(1-c\lambda_i)}$, where λ_i is the i^{th} largest eigen-value of $\tilde{\mathbf{W}}$.*

Proof: Taking the full eigen-value decomposition for $\tilde{\mathbf{W}}$:

$$\tilde{\mathbf{W}} = \sum_{i=1}^{n} \lambda_i \cdot u_i \cdot u_i^T = \mathbf{U}\mathbf{S}\mathbf{U}^T \tag{2.9}$$

where λ_i and u_i are the i^{th} largest eigen-value and the corresponding eigen-vector of $\tilde{\mathbf{W}}$, respectively. $\mathbf{U} = [u_1, ...u_n]$, and $\mathbf{S} = diag(\lambda_1, ..., \lambda_n)$. We have:

$$\begin{aligned} \tilde{\mathbf{\Lambda}} &= (\mathbf{S}^{-1} - c\mathbf{U}^T\mathbf{U})^{-1} \\ &= \sum_{i=1}^{n} \frac{\lambda_i}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \end{aligned} \tag{2.10}$$

By Lemma 2, we have:

$$\begin{aligned} \vec{r} &= (1-c)\sum_{i=1}^{n} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \cdot \vec{e_i} \\ \hat{\vec{r}} &= (1-c)\sum_{i=1}^{t} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \cdot \vec{e_i} \end{aligned} \tag{2.11}$$

Thus, we have

$$\begin{aligned} \|\vec{r} - \hat{\vec{r}}\|_2 &= \|(1-c)\sum_{i=t+1}^{n} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T \cdot \vec{e_i}\|_2 \\ &\leq (1-c)\|\sum_{i=t+1}^{n} \frac{1}{(1-c\lambda_i)} \cdot u_i \cdot u_i^T\|_2 \cdot \|\vec{e_i}\|_2 \\ &= (1-c)\sum_{i=t+1}^{n} \frac{1}{(1-c\lambda_i)} \end{aligned} \tag{2.12}$$

which completes the proof of Lemma 4. □

[6] Here, we ignore the low script i of $\vec{r}$ and $\hat{\vec{r}}$ for simplicity

Table 2.6: Summary of data sets

dataset	number of nodes	number of edges
CoIR	$5K$	$\approx 774K$
CoMMG	$\approx 52K$	$\approx 354K$
AP	$\approx 315K$	$\approx 1,834K$
AC	$\approx 291K$	$\approx 661K$

2.5 Experimental Results

2.5.1 Experimental Setup

In this Section, we present the experimental results, which are designed to answer the following questions: how does the proposed algorithms balance between approximation quality, pre-computational cost and on-line response time?

Data Sets

CoIR. This data set contains 5,000 images. The images are categorized into 50 groups, such as beach, bird, mountain, jewelry, sunset, etc. Each of the categories contains 100 images of essentially the same content, which serve as the ground truth. This is a widely used data set for image retrieval. Two kinds of low-level features are used, including color moment and pyramid wavelet texture feature. We use exactly the same method as in [HLZ+04] to construct the weighted graph matrix $\mathbf{W}$, which contains $5,000$ nodes and $\approx 774K$ edges

CoMMG. This data set is used in [PYFD04], which contains around 7,000 captioned images, each with about 4 captioned terms. There are in total 160 terms for captioning. In our experiments, 1,740 images are set aside for testing. The graph matrix $\mathbf{W}$ is constructed exactly as in [PYFD04], which contains $54,200$ nodes and $\approx 354K$ edges.

AP. The author-paper information of DBLP data set [7] is used to construct the weighted graph $\mathbf{W}$ as in equation (2.5): every author is denoted as a node in $\mathbf{W}$, and the edge weight is the number of co-authored papers between the corresponding two authors. On the whole, there are $\approx 315K$ nodes and $\approx 1,834K$ non-zero edges in $\mathbf{W}$.

AC. The author-conference information of DBLP data set is used to construct the bipartite graph $\mathbf{M}$: each row corresponds to an author and each column corresponds to a conference; and the edge weight $\mathbf{M}_{i,j}$ is the number of papers that the i^{th} author publishes in j^{th} conference. On the whole, there are $\approx 291K$ nodes ($\approx 288K$ authors and $\approx 3K$ conferences) and $\approx 661K$ non-zero edges in $\mathbf{M}$.

All the above data sets are summarized in table 2.6:

[7] `http://www.informatik.uni-trier.de/~ley/db/`

Table 2.7: Summary of typical applications with different datasets

	CBIR	CMCD	Ceps	NF
CoIR	$\surd$			$\surd$
CoMMG		$\surd$		
AP			$\surd$	
AC				$\surd$

Applications

As mentioned before, many applications can be built upon random walk with restart. In this chapter, we test the following applications:

- Center-Piece subgraph discovery (*CePS*) [TF06]
- Content based image retrieval (CBIR) [HLZ+04]
- Cross-modal correlation discovery (CMCD), including automatic captioning of images [PYFD04]
- neighborhood formulation (NF) for both uni-partite graph and bipartite graph [SQCF05]

The typical data sets for these applications in the past years are summarized in table 2.5.1.

Parameter Setting

The proposed methods are compared with OnTheFly, PreCompute and Blk. All these methods share 3 parameters: c, m and ξ_1. we use the same parameters for CBIR as [HLZ+04], that is $c = 0.95$, $m = 50$ and $\xi_1 = 0$. For the rest applications, we use the same setting as [PYFD04] for simplicity, that is $c = 0.9$, $m = 80$ and $\xi_1 = 10^{-8}$.

For B_LIN and NB_LIN, we take $\xi_2 = 10^{-4}$ to sparsify $\mathbf{Q}_1$, $\mathbf{U}$, and $\mathbf{V}$ which further reduces storage cost. We evaluate different choices for the remaining parameters. For clarification, in the following experiments, B_LIN is further referred as B_LIN(k, t, Eig/Part), where k is the number of partition, t is the target rank of the low-rank approximation, and "Eig/Part" denotes the specific method for doing low-rank approximation – "Eig" for eigen-value decomposition and "Part" for partition-based low-rank approximation. Similarly, NB_LIN is further referred as NB_LIN(t, Eig/Part), and Blk is further referred as Blk(k).

For the data sets with ground truth (*CoIR* and *CoMMG*), we use the relative accuracy $RelAcu$ as the evaluation criterion:

$$RelAcu = \frac{\widehat{Acu}}{Acu} \tag{2.13}$$

where $\widehat{Acu}$ and Acu are the accuracy values by the evaluated method and by PreCompute, respectively.

Another evaluation criterion is $RelScore$,

$$RelScore = \frac{\widehat{tScr}}{tScr}, \tag{2.14}$$

where $\widehat{tScr}$ and $tScr$ are the total relevance scores captured by the evaluated method and by PreCompute, respectively.

All the experiments are performed on the same machine with 3.2GHz CPU and 2GB memory.

2.5.2 *CoIR* Results

100 images are randomly selected from the original dataset as the query images and the precision vs. scope is reported. The user feedback process is simulated as follows. In each round of relevance feedback (RF), 5 images that are most relevant to the query based on the current retrieval result are fed back and examined. It should be pointed out that the initial retrieval result is equivalent to that for neighborhood formulation (NF). $RelAcu$ is evaluated on the first 20 retrieved images, that is, the precision within the first 20 retrieved images. In figure (2.3) and figure (2.4), the results are evaluated from three perspectives: accuracy vs. query time (QT), accuracy vs. pre-computational time (PT) and accuracy vs. pre-storage cost (PS). In the figure, the QT, PT and PS costs are in log-scale. Note that pre-computational time and storage cost are the same for both initial retrieval and relevance feedback, therefore, we only report accuracy vs. pre-computational time and accuracy vs. pre-storage cost for initial retrieval.

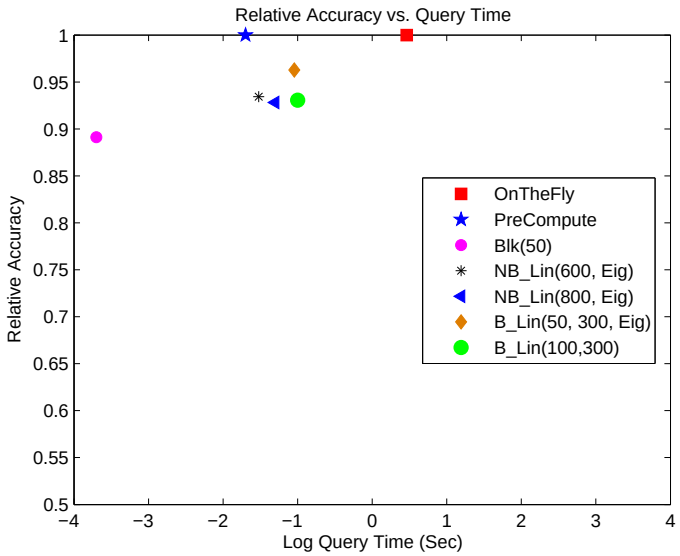
It can be seen that in all the figures, B_LIN and NB_LIN always lie in the upper-left zone, which indicates that the proposed methods achieve a good balance between on-line response quality and off-line processing cost. Both B_LIN and NB_LIN 1) achieve about one order of magnitude speedup (compared with OnTheFly); and 2) save one order of magnitude on pre-computational and storage cost. For example, B_LIN(50, 300, Eig) preserves 95%+ accuracy for both initial retrieval and relevance feedback, while it 1) achieves 32x speedup for on-line response (0.09Sec/2.91Sec), compared with OnTheFly; and 2)save 8x on storage (21M/180M) and 161x on pre-computational cost (90Sec/14,500Sec), compared with PreCompute. NB_LIN(600,Eig) preserves 93%+ accuracy for both initial retrieval and relevance feedback, while it 1) achieves 97x speedup for on-line response (0.03Sec/2.91Sec), compared with OnTheFly; and 2)saves 10x on storage(17M/180M) and 48x on pre-computational cost (303Sec/14,500Sec), compared with PreCompute.[8].

For the task of neighborhood formation (NF), figure (2.5) shows the result of RelScore vs. scope. It can been seen that by exploring both the block-wise and linear correlations structure simultaneously, 1) both Blk(50) and NB_LIN(50, Eig) capture most neighborhood information (for example, they both capture about 90% score for the precision on the first 10 retrieved images), and 2) B_LIN(50, 300, Eig) captures 95%+ score over the whole scope. (The improvement becomes even more significant with the increase of the scope).

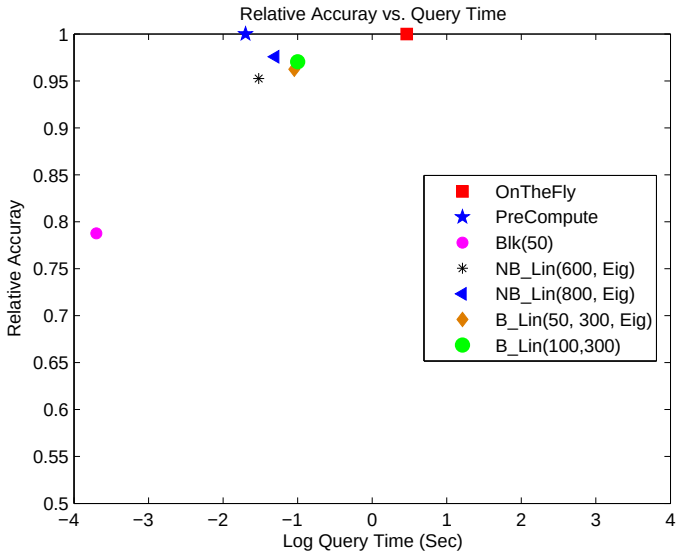
2.5.3 *CoMMG* Results

For this data set, we only compare NB_LIN with OnTheFly and PreCompute. The results are shown in figure (2.6) and figure (2.7). The x-axis of figure (2.6) and figure (2.7) is plotted in log-scale. Again, NB_LIN lies in the upper-left zone in all the figures, which means that

[8]We also perform experiment on BlockRank [KHMG03]. However, the result is similar with OnTheFly. Thus, we do not present it in this chapter.



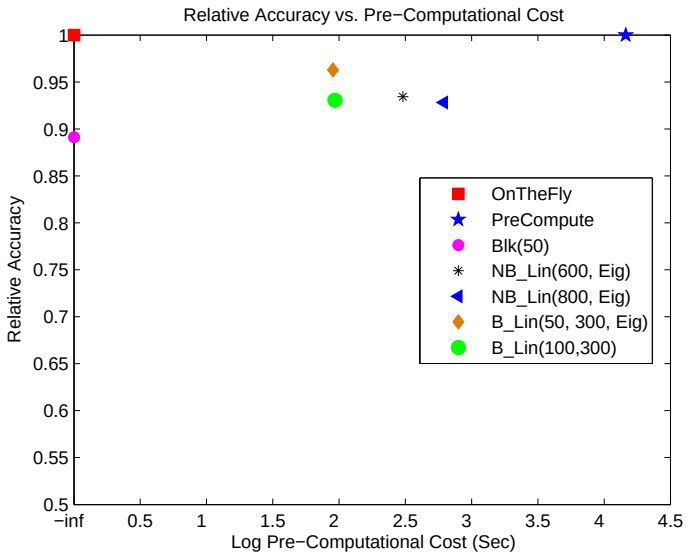
(a) Accuracy (Initial) vs. Log QT



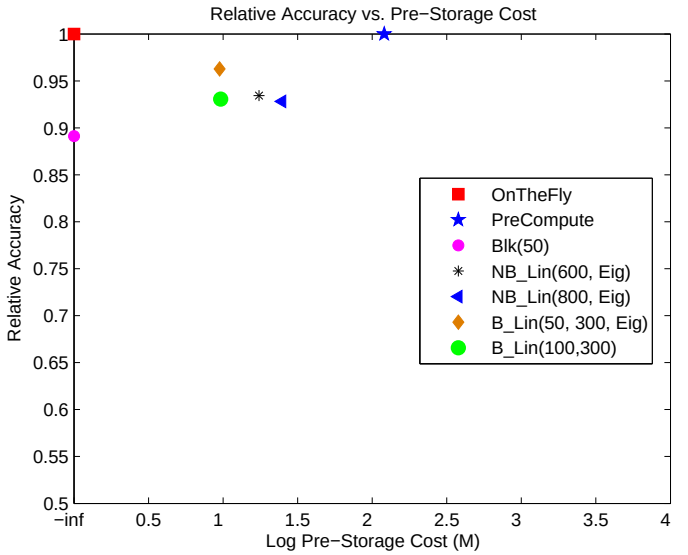
(b) Accuracy (RF) vs. Log QT

Figure 2.3: Evaluation on *CoIR* data set for CBIR. Accuracy vs. on-line cost. The proposed methods achieves a good balance between pre-computational cost, accuracy and on-line response time.

NB_LIN achieves a good balance between on-line quality and off-line processing cost. For example, NB_LIN(100, Eig) preserves 91.3% quality, while it 1) achieves 154x speedup for on-line response (0.029/4.50Sec), compared with OnTheFly; 2) saves 868x on storage (281/243,900M) and 479x on pre-computational cost (46/21,951Sec), compared with PreCompute. The relative precision/recall vs. scope is shown in figure (2.8).



(a)Accuracy (Initial) vs. Log PT



(b) Accuracy (Initial) vs. Log PS

Figure 2.4: Evaluation on *CoIR* data set for CBIR. Accuracy vs. off-line cost. The proposed methods achieves a good balance between pre-computational cost, accuracy and on-line response time.

2.5.4 *AP* Results

This dataset is used to evaluate *CePS* as in [TF06]. B_LIN is used to generate 1000 candidates, which are further fed to the original Ceps Algorithm [TF06] to generate the final center-piece subgraphs. We fix the number of query nodes to be 3 and the size of the subgraph to be 20.

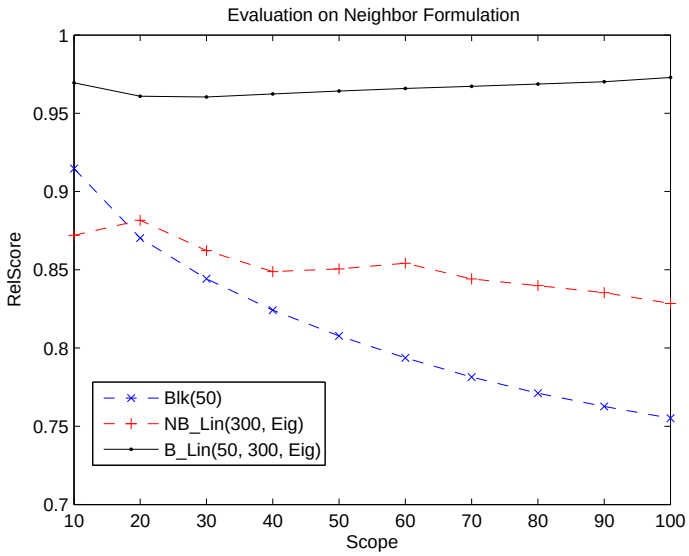


Figure 2.5: Evaluation on *CoIR* data set for NF. x-axis is the scope and y-axis is the normalized accuracy. Higher is better. The proposed B_LIN is best.

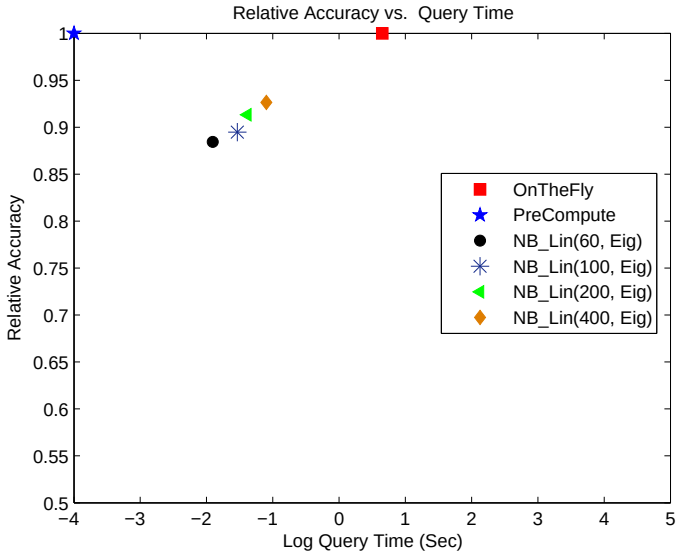
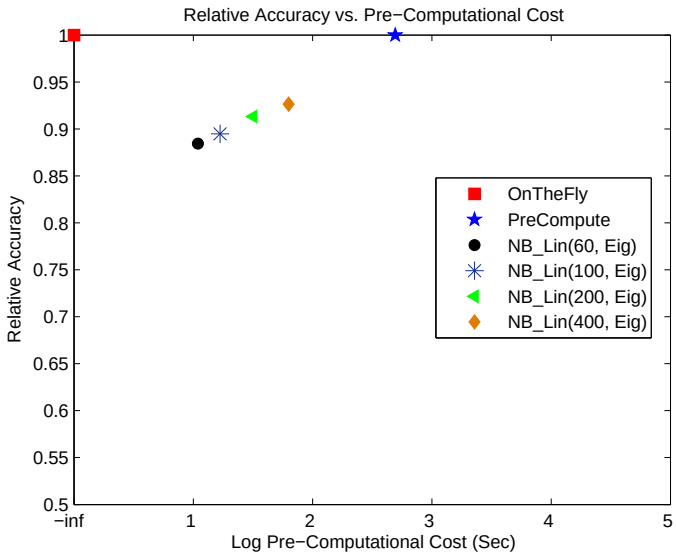


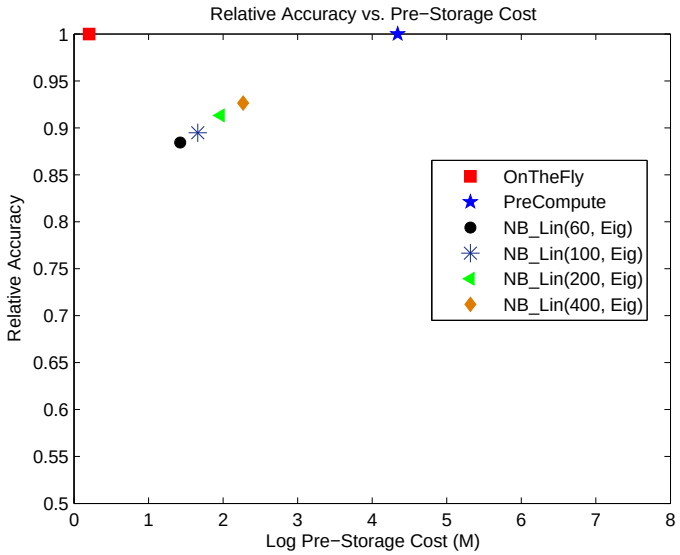
Figure 2.6: Evaluation on *CoMMG* data set for CMCD. Accuracy vs. on-line cost. The proposed B_LIN achieves a good balance between accuracy vs. pre-computational and query time.

RelScore is measured by ”Important Node Score” as in [TF06]. The result is shown in figure (2.9) and figure (2.10).

Again, B_LIN lies in the upper-left zone in all the figures, which means that B_LIN achieves a good balance between on-line quality and off-line processing cost. For example, B_LIN(100, 4000,



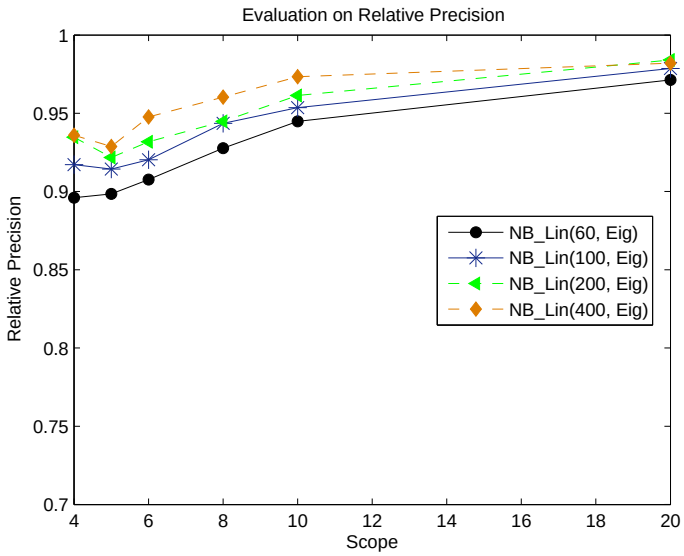
(a) Accuracy vs. Log PT



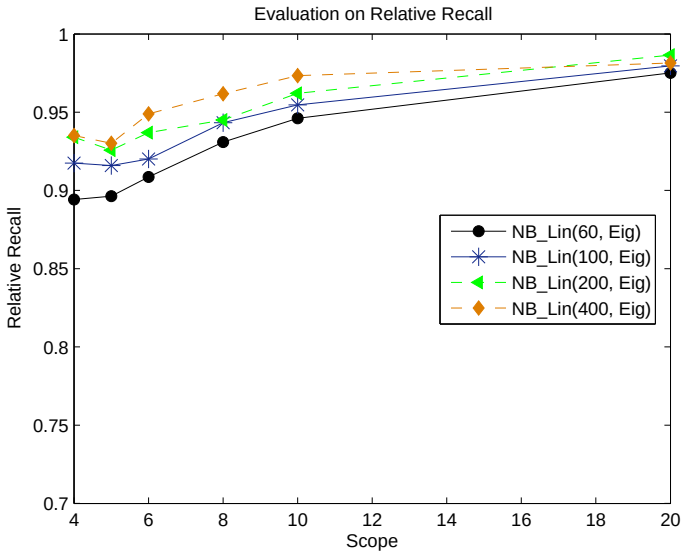
(b) Accuracy vs. Log PS

Figure 2.7: Evaluation on *CoMMG* data set for CMCD. Accuracy vs. off-line cost. The proposed B_LIN achieves a good balance between accuracy vs. pre-computational and query time.

Part) preserves 98.9% quality, while it 1) achieves 27x speedup for on-line response (9.45/258.2Sec), compared with OnTheFly; 2) saves 2264x on storage (269/609,020M) and 214x on pre-computational cost (8.7/1875Hour), compared with PreCompute.



(a) Relative precision



(b) Relative recall

Figure 2.8: Precision/recall for CMCD. x-axis is the scope and y-axis is the precision/recall. Higher is better.

2.5.5 *AC* Results

For this data set, the number of conferences ($3K$) is much less than that of the authors ($228K$). We evaluate BB_LIN for the following four tasks:

- **C_C:** Given a conference, find its most related conferences
- **C_A:** Given a conference, find its most related authors

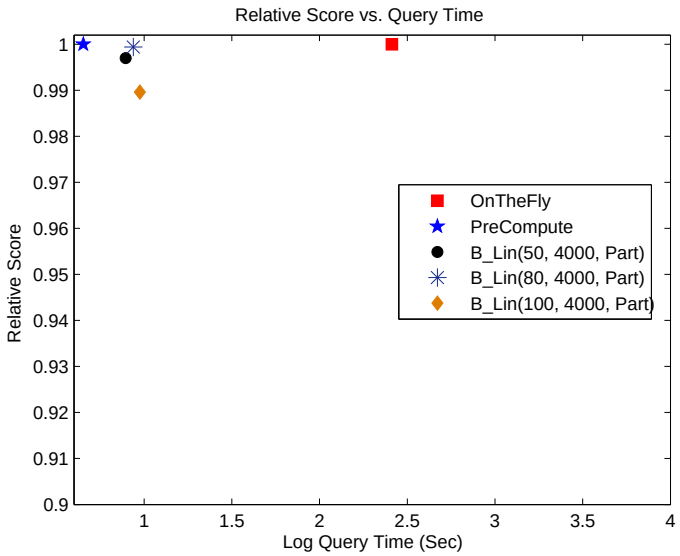


Figure 2.9: Evaluation on *AP* data set for CePS. Accuracy vs. on-line cost. The proposed B_LIN achieves a good balance between accuracy vs. pre-computational and query time.

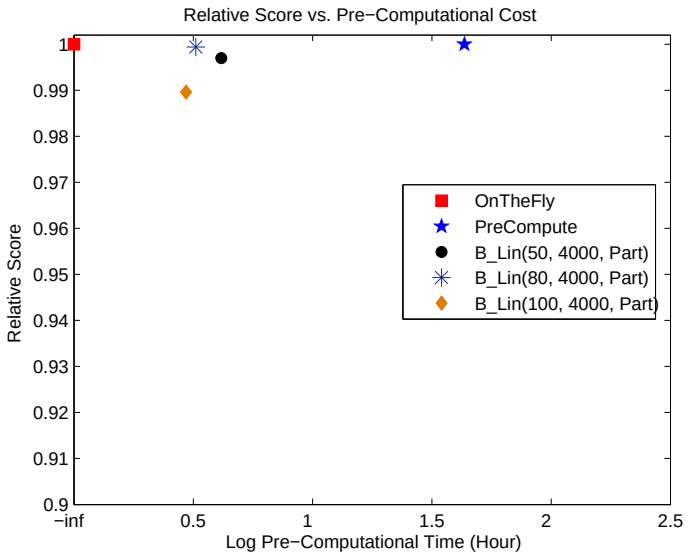
Table 2.8: Evaluation on *AC*for NF

Method	QT(Sec)	PT(Sec)	PS(M)
OnTheFly	23.97	0	6.7
PreCompute	0.001	6,990,648	626,250
BB_LIN(C, A)	0.097	20.50	56
BB_LIN(C, C)	0.013	20.50	56
BB_LIN(A, C)	0.035	20.50	56
BB_LIN(A, A)	0.13	20.50	56

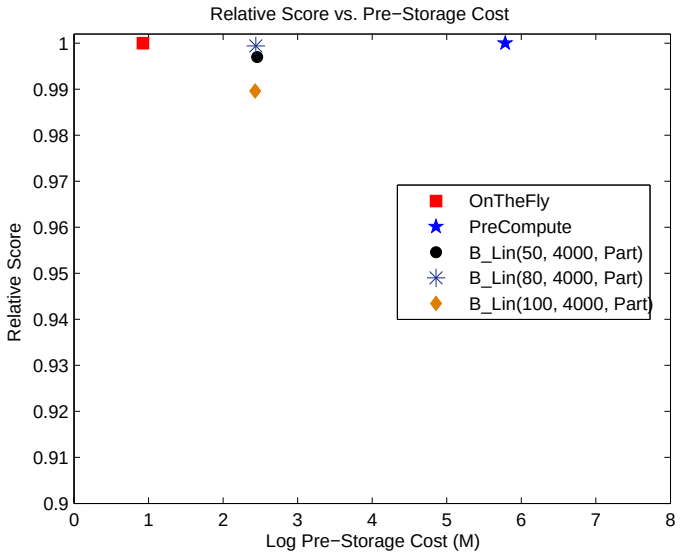
- **A_A:** Given an author, find its most related authors
- **A_C:** Given an author, find its most related conferences

On this application, BB_LIN preserves 100% accuracy for all the tasks. Thus, in table (2.8), we only report Query time (QT), Pre-computational time (PT), and Pre-storage cost (PS). Note that the query time for BB_LIN might differ for the different tasks. For clarification, BB_LIN is further referred as BB_LIN(C/A C/A). (For example, BB_LIN(C, A) denotes using BB_LIN for C_A task.)

As shown in table (2.8), BB_LIN can achieve up to 3 orders of magnitude speedup, with light off-line computational and storage cost (20.5Sec for pre-computation and 56M for pre-storage). For example, it achieves 180x speedup for **A_A** (0.13/23.98Sec) and 1,800 speedup for **C_C**(0.013/23.98Sec).



(a) Accuracy vs. Log PT



(b) Accuracy vs. Log QS

Figure 2.10: Evaluation on *AP* data set for CePS. Accuracy vs. off-line cost. The proposed B_LIN achieves a good balance between accuracy vs. pre-computational and query time.

2.6 Related Work

In this Section, we briefly review related work, which can be categorized into three groups: (1) random walk related methods; (2) graph partitioning methods and (3) the methods for low-rank approximation.

Random walk related methods. There are several methods similar to RWR, including electricity-

based method [ZGL03], graph-based Semi-supervised learning [ZBL+03] [FMT04] and so on. Exact solution of these methods usually requires the inversion of a matrix which is often diagonal dominant and of big size. Other methods sharing this requirement include regularized regression, Gaussian process regression [RW06], and so on. Existing fast solutions for RWR include Hub-vector decomposition based [JW03]; block structure based [KHMG03] [SQCF05]; fingerprint based [FR04], and so on. Many applications take random walk and related methods as the building block, including PageRank [PBMW98], personalized PageRank [Hav02], SimRank [JW02], neighborhood formulation in bipartite graphs [SQCF05], content-based image retrieval [HLZ+04], cross modal correlation discovery [PYFD04], the BANKS system [ABC+02], ObjectRank [BHP04], RelationalRank [GMT04], and so on.

Graph partition and clustering. Several algorithms have been proposed for graph partition and clustering, e.g. METIS [KK99], spectral clustering [NJW01], flow simulation [FLG00], co-clustering [DMM03], and the betweenness based method [GN]. It should be pointed out that the proposed method is orthogonal to the partition method.

Low-rank approximation: One of the widely used techniques is singular vector decomposition (SVD) [GL96], which is the base for a lot of powerful tools, such as latent semantic index (LSI) [DDL+90], principal component analysis (PCA) [Jol02], and so on. For symmetric matrices, a complementary technique is the eigen-value decomposition [GL96]. More recently, CUR decomposition has been proposed for sparse matrices [AM01].

2.7 Conclusions and Discussions

Summary of This Chapter. In this chapter, we introduce random walk with restart as a proximity measurement, and propose a fast solution for it. The main contributions of this chapter are as follows:

- The design of B_LIN and its derivative, NB_LIN. These methods take advantages of the block-wise structure and linear correlations in the adjacency matrix of real graphs, using the Sherman-Morrison Lemma.
- The proof of an error bound for NB_LIN. To our knowledge, this is the first attempt to derive an error bound for fast random walk with restart.
- Extensive experiments are performed on several real datasets, on typical applications. The results demonstrate that our proposed algorithm can nicely balance the off-line processing cost and the on-line response quality. In most cases, our methods preserve 90%+ quality, with dramatic savings on the pre-computation cost and the query time.
- A fast solution (BB_LIN) for one particular class of bipartite graphs. Our method achieves up to 1,800x speedup with light pre-computational and storage cost, without suffering quality loss.

Discussions. In [TFGER07], we also explored another proximity definition (*DAP*) in order to leverage the edge directionality, which is based on escape probability augmented with a universal sink. There, we also two fast solutions in two different settings. We also generalized our definitions to group proximity (to quantify how close two groups nodes are). It is interesting to point out that

our analysis shows that *DAP* can be actually based on random walk with restart. It is worth pointing out that in some specific scenarios/applications, we can often do better by leveraging the special properties coded by that specific applications. We will present the details in the following few chapters (chapters 3-6 and chapter 9).

Part II

Querying Static Graphs

Chapter 3

Case Study #1: Center-Piece Subgraphs

Summary of This Chapter

- **Questions we want to answer:**

  Q: Given Q query nodes in a social network (e.g., co-authorship network), how to find the node(s) and the resulting subgraph, that have strong connections to all or most of the Q query nodes?

- **Our answers and contributions**

  A1: We formally formulate the problem (Center-Piece Subgraph Discovery).

  A2: We proposed an effective and efficient algorithm to find CePS.

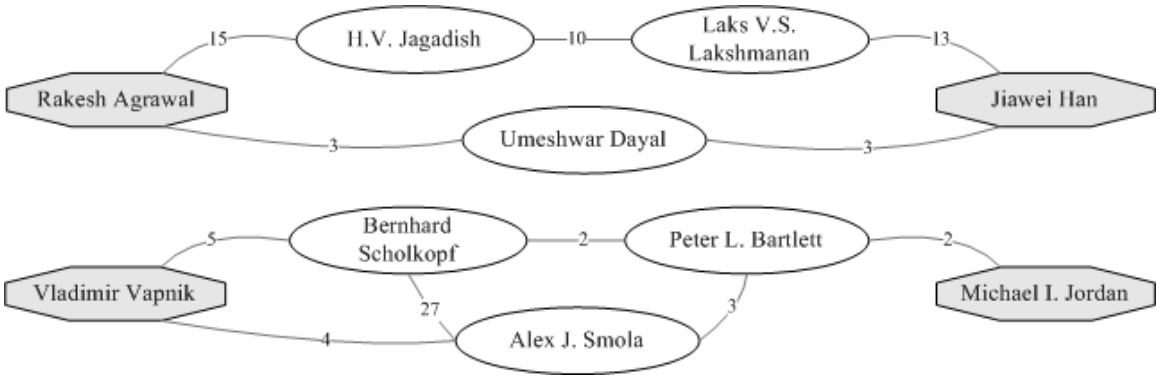
3.1 Introduction

Graph mining has been attracting increasing interest recently, for community detection, partitioning, frequent subgraph discovery and many more. Here we introduce and solve a novel problem, the "*Center-Piece Subgraph*" (*CePS*) problem: Given Q query nodes in a social network (e.g., co-authorship network), find the node(s) and the resulting subgraph, that have strong connections to all or most of the Q query nodes. The discovered nodes could contain a common advisor, or other members of the research group, or an influential author in the research area that the Q nodes belong to. There are multiple alternative applications, e.g., law enforcement, gene regulatory networks.

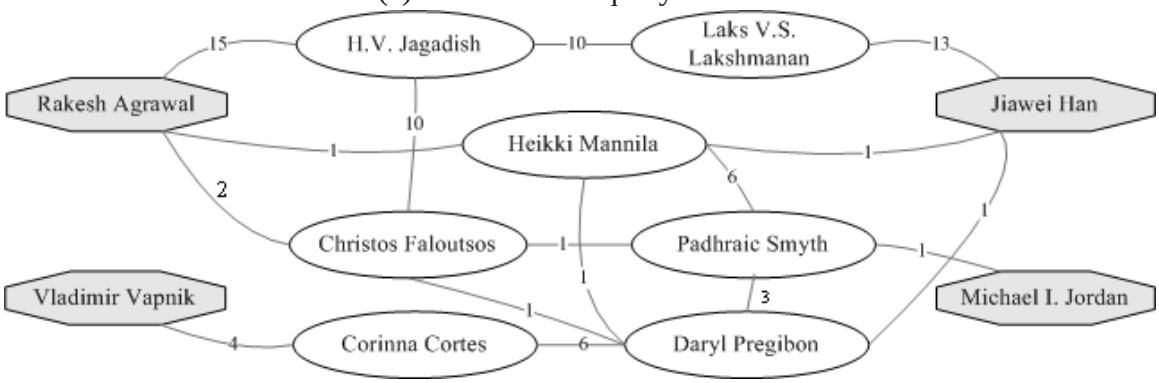
Earlier work [FMT04] focused on the so-called "connection subgraphs". Although the inspiration for the current work, the connection subgraph algorithm can only handle the case of Q=2. This is exactly the major contribution of our work: we allow not only pairs of query nodes, but any arbitrary number Q of them.

Figure 3.1 gives screenshots of our system, showing our solution on a DBLP graph, with Q=4 query nodes. All 4 researchers are in data mining, but the first two (Rakesh Agrawal and Jiawei Han) are more on the database side, while Michael Jordan and Vladimir Vapnik are more on the

machine learning and statistical side. Figure 3.1(b) gives our *CePS* subgraph, when we request nodes with strong ties to all four query nodes. The results make sense: researchers like Daryl Pregibon, Padhraic Smythe and Heikki Mannila are vital links, because of their cross-disciplinarity and their strong connections with both the above sub-areas. Figure 3.1(a) illustrates an important aspect of our work, the $K\_softAND$ feature, which we will discuss very soon. In a nutshell, in a $K\_softAND$ query, our method finds nodes with connections to at least k of the query nodes ($k = 2$ in Figure 3.1(a)).



(a) " K_softAND query": $k = 2$



(b) " AND query"

Figure 3.1: Center-piece subgraph among Rakesh Agrawal, Jiawei Han, Michael I. Jordan and Vladimir Vapnik.

Thus, we define the Center-Piece Subgraph problem, as follows:

Problem 1. *Center-Piece Subgraph Discovery(*CePS*)*

Given: *an edge-weighted undirected graph* $\mathbf{W}$, Q *nodes as source queries* $\mathcal{Q} = \{q_i\}$ $(i = 1, ..., Q)$, *the softAND coefficient* k *and an integer budget* b

Find: *a suitably connected subgraph* $\mathcal{H}$ *that (a) contains all query nodes* q_i *(b) at most* b *other vertices and (c) it maximizes a "closeness" function* $g(\mathcal{H})$.

By problem 1, there are three requirements in *CePS*: (a) the resulting subgraph is small (with less or equal than b nodes); (b) the subgraph is reasonably connected ("connection") and (c) the nodes in the resulting subgraph are close to the query set (the "closeness"). We will give the

detailed definitions of "connection" and "closeness" later in the chapter.

Allowing Q query nodes creates a subtle problem: do we want the qualifying nodes to have strong ties to all the query nodes? to at least one? to at least a few? We handle all of the above cases with our proposed $K\_softAND$ queries. Figure 3.1(a) illustrates the case where we want intermediate nodes with good connections to at least $k = 2$ of the query nodes. Notice that the resulting subgraph is much different now: there are two disconnected components, reflecting the two sub-communities (databases/statistics).

The contributions of this work are the following

- The problem definition, for arbitrary number Q of query nodes, with careful handling of a lot of the subtleties.
- The introduction and handling of $K\_softAND$ queries.
- *EXTRACT*, a novel subgraph extraction algorithm.
- The design of a fast, approximate method, which provides a $6:1$ speedup with little loss of accuracy.

The system is operational, with careful design and numerous optimizations, like alternative normalization of the adjacency matrix, a fast algorithm to compute the scores for $K\_softAND$ queries.

Our experiments on a large real data set (DBLP) show that our method returns results that agree with our intuition, and that it can be made fast (a few seconds response time), while retaining most of the accuracy (about 90%).

The rest of this chapter is organized as follows: Section 3.2 provides an overview of the proposed method: *CePS*. The closeness score calculation is proposed Section 3.3 and its variants are presented in the Appendix. The "*EXTRACT*" algorithm and the speeding up strategy are provided in Section 3.4 and Section 3.5, respectively. We present experimental results in Section 3.6 and we review some related work in Section 3.7. Finally, we conclude the chapter in Section 3.8.

3.2 Proposed Method: Overview

Given the budget b, we want to find a subgraph which (a) is reasonably connected ("connection") and (b) the nodes in this subgraph are close wrt the query set ("closeness").

For the "closeness" requirement, we want to find a subgraph $\mathcal{H}$ which is close wrt the query set. To this end, let us first define the closeness score for a single node in this subgraph $\mathcal{H}$. More specifically, for a given node j in $\mathcal{H}$, we have two types of closeness scores:

- Let $r(i, j)$ be the closeness score of a given node j wrt the query q_i;
- Let $r(\mathcal{Q}, j)$ be the closeness score of a given node j wrt the query set $\mathcal{Q}$.

A natural way to measure the closeness of the subgraph $\mathcal{H}$ wrt the query set is to measure the closeness of the nodes it contains: the more close nodes (wrt the source queries) it contains, the better (in terms of closeness) $\mathcal{H}$ is. Thus, the goodness criterion in terms of closeness of $\mathcal{H}$ can be defined as:

$$g(\mathcal{H}) = \sum_{j \in \mathcal{H}} r(\mathcal{Q}, j) \tag{3.1}$$

By eq. 3.1, a subgraph is good in terms of closeness if $g(\mathcal{H})$ is high. With the above criterion, a straightforward way to choose the "best" (in terms of closeness) subgraph should be the one which maximizes $g(\mathcal{H})$:

$$\mathcal{H}^* = argmax_{\mathcal{H}} g(\mathcal{H}) \tag{3.2}$$

However, no connection is guaranteed in this way and the resulting subgraph $\mathcal{H}$ might be a collection of isolated nodes. Thus, there are two basic problems in center-piece subgraph discovery: (1) how to define a reasonable closeness score $r(\mathcal{Q}, j)$ for a given node j; (2): how to quickly find a connection subgraph maximizing $g(\mathcal{H})$. Moreover, since it might be very difficult to directly calculate the closeness score $r(\mathcal{Q}, j)$, we further decompose it into two steps. The pseudo code for the proposed method (*CePS*) is listed as follows:

Table 3.1: Overview of *CePS*

Input: the weighted graph $\mathbf{W}$, the query set $\mathcal{Q}$, $K\_softAND$ coefficient k and the budget b
Output: the resulting subgraph $\mathcal{H}$
Step 1: Individual Score Calculation. Calculate the closeness score $r(i, j)$ for a single node j wrt a single query node q_i
Step 2: Combining Individual Scores. Combine the individual score $r(i, j)$ to get the closeness score $r(\mathcal{Q}, j)$ for a single node j wrt the query set $\mathcal{Q}$
Step 3: *"EXTRACT".* Extract quickly a connection subgraph $\mathcal{H}$ with budget b maximizing the closeness criteria $g(\mathcal{H})$

3.3 Closeness Score Calculation for a Single Node

In this Section, we deal with the closeness score calculation for a single node. That is, how to define the closeness score of a given node wrt the query set. For clarification, whenever we say that a node is 'good' in this Section, we mean that this node is 'good' in term of closeness. Also, we use the terms "goodness" and "closeness" interchangeably in this Section.

There are two basic concepts in closeness score calculation:

- Let $r_{i,j}$ be the *steady-state probability* that a particle will find itself at node j, when it does random walk with restarts (RWR) from query node q_i.
- Let $r(\mathcal{Q}, j, k)$ be the *meeting probability*, that is, the steady-state probability that at least k-out-of-Q particles, doing RWR from the query nodes of $\mathcal{Q}$, will all find themselves at node j in the steady state; k is the K_softAND coefficient.

These two kinds of steady probability ($r_{i,j}$ and $r(\mathcal{Q}, j, k)$) are the base of our closeness score calculation (for both $r(i, j)$ and $r(\mathcal{Q}, j)$). It's basic idea is that: suppose there are Q random particles doing RWR from each query node independently; then after convergency, each particle has some *steady-state probability* staying at the node j; and different particles have some *meeting probability* at the node j. The *steady-state probability* and the *meeting probability* provide some hints on how the node j is related with the source queries, and are used to compute the closeness

Table 3.2: Symbols

Symbol	Description
N	total number of nodes in the weighted graph
m	iteration step
c	fly-out probability for random walk with restart
$\vec{e}_i$	$N \times 1$ unit query vector, with all zeros except one at row q_i
$\mathbf{W} = \{w_{i,j}\}$	the edge weighted matrix ($i, j = 1, ..., N$)
$\mathbf{D} = \{d_{i,j}\}$	$N \times N$ matrix, $d_{i,i} = d_i$, and $d_{i,j} = 0$ for $i \neq j$
d_i	the sum of the i^{th} row of W
$\mathcal{H}$	the chosen center-piece subgraph
Q	number of source query nodes
$\mathcal{Q} = \{q_i\}$	set of query nodes ($i = 1, ..., Q$)
$\acute{\mathcal{Q}}$	the first $(Q-1)$ query nodes of query set $\mathcal{Q}$, $\acute{\mathcal{Q}} = \{q_i\}, (i = 1, .., (Q-1))$
$\varnothing$	null query set, which contains no query node
$r(i, j)$	goodness score for a single node j wrt query node q_i
$r(\mathcal{Q}, j)$	goodness score for a single node j wrt query set $\mathcal{Q}$
$r(\mathcal{Q}, (j, l))$	goodness score for a single edge (j, l) wrt query set $\mathcal{Q}$
$r_{i,j}$	*steady-state probability* of a single node j wrt query node q_i
$\mathbf{R}$	$Q \times N$ matrix of $[r_{i,j}]$
$r(\mathcal{Q}, j, k)$	*meeting probability* of a single node j, wrt $k (k = 1, .., Q)$ or more of the query nodes of $\mathcal{Q}$
$r(i, (j, l))$	*meeting probability* of a single edge (j, l), wrt query node q_i
$r(\mathcal{Q}, (j, l), k)$	*meeting probability* of a single edge (j, l), wrt $k (k = 1, .., Q)$ or more of the query nodes of $\mathcal{Q}$

score of node j. Moreover, by designing different *meeting probability*, we can get the specific type of closeness score tailored for the specific query scenario. Table 3.2 lists all the symbols and definitions used throughout this chapter.

3.3.1 Individual score calculation

Here we want to compute the closeness score $r(i, j)$ of a single node j, for a single query node q_i. We propose to use random walks with restart, from the query node q_i.

Suppose a random particle starts from query q_i, the particle iteratively transmits to its neighborhood with the probability that is proportional to the edge weight between them, and also at each step, it has some probability c to return to node q_i. $r(i, j)$ is defined as the *steady-state probability* $r_{i,j}$ that the particle will finally state at node i:

$$r(i, j) \triangleq r_{i,j} \tag{3.3}$$

More formally, if we put all the $r_{i,j}$ probabilities into matrix form $\mathbf{R} = [r_{i,j}]$, then

$$\mathbf{R}^T = c\mathbf{R}^T \times \tilde{\mathbf{W}} + (1-c)\mathbf{E} \tag{3.4}$$

where $\mathbf{E} = [\vec{e}_i](i = 1, ..., Q)$ is the $N \times Q$ matrix, $(1-c)$ is the fly-out probability, and $\tilde{\mathbf{W}}$ is the adjacency matrix $\mathbf{W}$ appropriately normalized, say, column-normalized:

$$\tilde{\mathbf{W}} = \mathbf{W} \times \mathbf{D}^{-1} \tag{3.5}$$

The problem can be solved in many ways - we choose the iteration method, iterating Eq. 3.4 until convergence. For simplicity, in this chapter, we iterate Eq. 3.4 m times, where m is a pre-fixed iteration number.

3.3.2 Combining individual scores

Here we want to combine the individual score $r(i,j)(i = 1, ..., Q)$ to get $r(\mathcal{Q}, j)$, the closeness score for a single node j wrt the query set $\mathcal{Q}$. We propose to use the *meeting probability* $r(\mathcal{Q}, j, k)$ of random walk with restart. Furthermore, by using different softAND coefficient k, we can deal with different types of query scenario.

The most common query scenario might be that "given Q query nodes, find the subgraph $\mathcal{H}$ the nodes of which are important/good wrt ALL queries". In this case, $r(\mathcal{Q}, j)$ should be high if and only if there is a high probability that ALL particles will finally meet at node j:

$$r(\mathcal{Q}, j) \triangleq r(\mathcal{Q}, j, Q) = \prod_{i=1}^{Q} r(i,j) \tag{3.6}$$

Eq. 3.6 actually defines a logic AND operation in terms of individual closeness scores: the node j is important wrt the query set $\mathcal{Q}$ if and only if it is important wrt every query node. Thus, we refer such query type as "AND query".

A complemental query scenario is "OR query": "given Q queries, find the subgraph $\mathcal{H}$ the nodes of which are important wrt at least ONE query". In this case, $r(\mathcal{Q}, j)$ should be high if and only if there is a high probability that at least one particle will finally stay at node j:

$$r(\mathcal{Q}, j) \triangleq r(\mathcal{Q}, j, 1) = 1 - \prod_{i=1}^{Q} (1 - r(i,j)) \tag{3.7}$$

Eq. 3.7 defines a logic OR operation in terms of individual importance scores: the node j is important wrt the source queries if and only if it is important wrt at least one source query.

Besides the above two typical scenarios, the user might also ask "given Q queries, find the subgraph $\mathcal{H}$ the nodes of which are important wrt at least $k(1 \leq k \leq Q)$ queries". We refer such query type as "$K\_softAND$ query". In this case, $r(\mathcal{Q}, j)$ should be high if and only if there is a high probability that at least k-out-of-Q particles will finally meet at node j.

$$r(\mathcal{Q}, j) \triangleq r(\mathcal{Q}, j, k) \tag{3.8}$$

To avoid exponential enumeration (which is $O(2^k)$), Eq. 3.8 can be computed in a recursive manner:

$$r(\mathcal{Q}, j, k) = r(\acute{\mathcal{Q}}, j, k-1) \cdot r(Q, j) + r(\acute{\mathcal{Q}}, j, k) \cdot (1 - r(Q, j)) \tag{3.9}$$

where $r(\mathcal{Q}, j, 1) = 1 - \prod_{i=1}^{Q} (1 - r(i, j))$.

Intuitively, Eq. 3.8 defines a logic operation in terms of individual importance scores that is between logic AND and logic OR. In this chapter, we refer it as logic K_softAND: the node j is important wrt the source queries if and only if it is important wrt at least k-out-of-Q source queries.

It is worth pointing out that both "AND query" and "OR query" can be viewed as special cases of "$K\_softAND$ query": "AND query" is actually "Q_softAND query"; while "OR query" is actually "1_softAND query"

3.3.3 Variation: normalization on W

To compute the closeness score $r(i, j)$ and $r(\mathcal{Q}, j)$, we need to construct the transition matrix $\tilde{\mathbf{W}}$ for random walk with restart. A direct way is to normalize $\mathbf{W}$ by column as Eq. 3.5. However, as pointed out in [FMT04], there might be the so called "pizza delivery person" problem, that is, the node with high degree is prone to receive too much attention (receiving too high individual closeness score in our case). To deal with this problem, we propose to normalize $\mathbf{W}$ as Eq. 3.10. The normalized weighted graph $\mathbf{W}$ will be further used to formulate the transition matrix $\tilde{\mathbf{W}}$ by Eq. 3.5.

$$w_{j,l} \leftarrow w_{j,l}/(d_j)^{\alpha} \tag{3.10}$$

for all $j, l = 1, ..., N$.

The motivation of normalization is as follows: for the high degree node j, every edge $(j, l)(l = 1, ..., N)$ is penalized by $(d_i)^{\alpha}$ and vice versa. The coefficient α control the penalization strength: bigger α indicates stronger penalization. Note that the idea of penalizing the node with high degree is similar with that of setting a universal sink node in [FMT04].

3.4 The "Extract" Algorithm

In this Section, we propose "EXTRACT" algorithm to deal with the "connection" requirement of *CePS*: what do we mean by "connection" and how to find the resulting subgraph which satisfies the connection requirement while maximizing the goodness/closeness with the limited budget b.

The "EXTRACT" algorithm takes as input the weighted graph $\mathbf{W}$, the importance scores on all nodes, the budget b and the softAND coefficient k; and produces as output a small, unweighted, undirected graph $\mathcal{H}$. The basic idea is similar with the display generation algorithm in [FMT04]: 1) instead of trying to find an optimal subgraph maximizing $g(\mathcal{H})$ directly, we decompose it into finding key paths incrementally; 2) by sorting the nodes in order, we can quickly find the key paths by dynamic programming in the acyclic graph.

However, we cannot directly apply the original display generation algorithm since it can only deal with pair source queries (and also the resulting subgraph is sensitive to the order of the source queries). To deal with this issue, we extend the original algorithm in the following aspects:

(1) Instead of finding a source-source path, at each step, the algorithm will pick up a most promising destination node pd; and try to find a source-destination path for each source query node.

(2) The order (which will be used in the dynamic programming) is specified with each source query node.

(3) Key path discovery differs with the different query types: for "AND query" the algorithm will discover Q paths for all source nodes at each step; for "K_softAND query", it only discovers k paths for the first k source nodes; while for "OR query", the algorithm will only find 1 path at each step.

Before presenting the algorithm, we require the following definitions:

- *SPECIFIED DOWNHILL NODE.* Node u is downhill from node v wrt source q_i ($v \rightarrow d_i, u$) if $r(i, v) > r(i, u)$;
- *SPECIFIED PREFIX PATH.* A specified prefix path $P(i, u)$ is any downhill path that starts from source q_i and ends at node u; that is, $P(i, u) = (u_0, u_1, ..., u_n)$ where $u_0 = q_i, u_n = u$, and $u_j \rightarrow d_i, u_{j+1}$;
- *EXTRACTED GOODNESS.* The extracted goodness is the total goodness score of the nodes within the subgraph $\mathcal{H}$: $CF(\mathcal{H}) = \sum_{j \in \mathcal{H}} r(\mathcal{Q}, j)$.
- *EXTRACTED MATRIX.* $C_s(i, u)$ is the extracted goodness score from source node q_i to node u along the prefix path $P(i, u)$ so that:
  1. $P(i, u)$ has exactly s nodes not in the present output graph $\mathcal{H}$
  2. $P(i, u)$ extracts the highest goodness score among all such paths that start from q_i and end at u.
- *ACTIVE SOURCE.* For $K\_softAND$, the source node q_i is active wrt destination node pd if $r(i, pd) \geq r^{(k)}(i, pd)$, where $r^{(k)}(i, pd)$ is the k^{th} largest value among $r(i, pd), (i = 1, ..., Q)$. Note that the number of active source differs with the query type[1]: for "OR query", there is only one active source while for "AND query", all sources are active. For a specific query type, an active source q_i might turn into inactive when the destination node pd changes and vice versa.

The destination node pd can be decided by Eq. 3.11:

$$pd = argmax_{j \notin \mathcal{H}} r(\mathcal{Q}, j) \tag{3.11}$$

where $\mathcal{H}$ is the partially built output subgraph.

In order to make the resulting subgraph to be "reasonably connected", we want to make sure that (1) there is at least one path that connects the destination node pd and each query node for AND query; and (2) there is at least one path that connects the destination node pd and k-out-of-Q query nodes. In this way, not only does the algorithm select good/close nodes wrt the query set (i.e., a destination node pd with high $r(\mathcal{Q}, j)$), but also it provides some interpretations on why such nodes are good/close wrt the query set.

[1]Since both "AND query" and "OR query" can be viewed as special cases of "K_softAND query", the number of active sources is actually k for all query types.

However, we do not want to find an arbitrary path to connect the destination node pd and the one query node since (1) we also want to make sure that the remaining nodes (besides the destination node pd) in the resulting subgraph are good/close wrt the query set; and (2) the number of total nodes in the resulting subgraph is limited by the budget b. Therefore, we aim to find a path from one query node and the destination node pd which maximizes the total captured combined scores along the path over the length of the path. Also, since we try to find the resulting subgraph gradually, a new path might include some existing nodes in the current subgraph. In order to encourage different paths to share with the same nodes since the budget b is limited, we define the length of the path is defined as the number of new nodes in this path.

In order to discover a new path between the source q_i and the promising node pd, we arrange the nodes in descending order of $r(i,j)(j = 1, ..., n)$: $\{u_1 = q_i, u_2, u_3, ..., pd = u_n\}$. (note that all nodes with smaller $r(i,j)$ than $r(i,pd)$ are ignored). Then we fill the extracted matrix C in topological order so that when we compute $C_s(t,u)$, we have already computed $C_s(t,v)$ for all $v \rightarrow d_i, u$. On the other hand, as the subgraph is growing, a new path may include nodes that are already present in the output subgraph, our algorithm will favor such paths as in [FMT04]. The complete algorithm to discover a single path from source node q_i and the destination node pd is given in table 3.3.

Table 3.3: Single Key Path Discovery

1. Let len be the maximum allowable path length
2. For $j \leftarrow [1, ..., n]$
   2.1. Let $v = u_j$
   2.2. For $s \leftarrow [2, ..., len]$
      If v is already in the output subgraph
         $s' = s$
      Else
         $s' = s - 1$
      Let $C_s(i,v) = max_{u|u \rightarrow d_i, v}(C_{s'}(i,u) + r(\mathcal{Q}, v))$
3. Output the path maximizing $C_s(i,pd)/s$, where $s \neq 0$

Based on the previous preparations, the *EXTRACT* algorithm can be given in table 3.4.

3.5 Speeding up *CePS*

To compute $r(i,j)$, we have to solve a linear system. When the data set is large (or more precisely, when the total number of the edges in the graph is large), the processing time could be long. Note that we can directly apply the proposed B_LIN in chapter 2 to compute $r(i,j)$. Here, we consider an alternative way to speed up the whole process.

Note that Eq. 3.4 can be solved in closed form:

$$\mathbf{R}^T = (1-c)(\mathbf{I} - c\tilde{\mathbf{W}})^{-1}\mathbf{E} \tag{3.12}$$

Table 3.4: Our *EXTRACT* Algorithm

1. Initialize output graph $\mathcal{H}$ null
2. Let len be the maximum allowable path length
3. While $\mathcal{H}$ is not big enough
   - 3.1. Pick up destination node pd by Eq. 3.11
   - 3.2. For each active source node q_i wrt node pd
     - 3.2.1. use table 3.3 to discover a key path $P(q_i, pd)$
     - 3.2.2. add $P(q_i, pd)$ to $\mathcal{H}$
4. Output the final $\mathcal{H}$

Thus, an obvious way to speed up *CePS* is to pre-compute and store the matrix $\mathbf{A} = (\mathbf{I} - c\tilde{\mathbf{W}})^{-1}$, then $\mathbf{R}^T = (1-c)\mathbf{AE}$ can be computed on-line nearly real-time. However, in this way, we have to store the whole $N \times N$ matrix A, which is a heavy burden when N is big.

As suggested by [SQCF05], the goodness score $r(i,j)(j = 1, ..., N)$ is very skewed, that is, most values of $r(i,j)$ are near zero and only a few nodes have high value. Based on this observation, we propose to pre-partition the original weighted graph $\mathbf{W}$ into several partitions and only use the partitions containing the source queries to run *CePS*. In this chapter, we use METIS [KK99] as the partition algorithm.

The pseudo code for the accelerated *CePS* is summarized as follows:

Table 3.5: Fast *CePS*

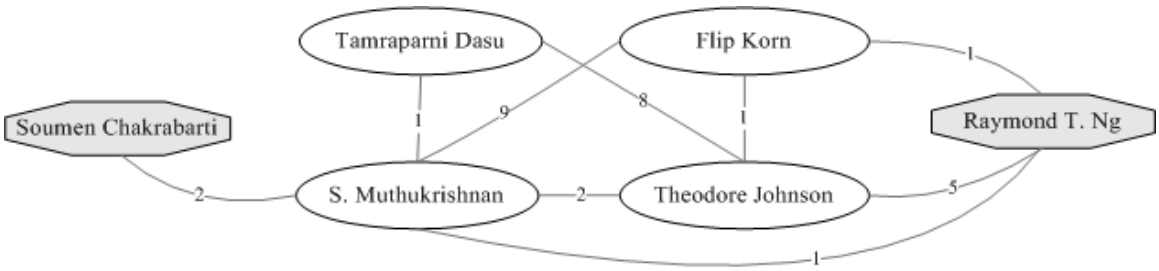
Input: the weighted graph $\mathbf{W}$, the query set $\mathcal{Q}$, $K\_softAND$ coefficient k, the budget b, and the number of partitions p;
Output: the resulting subgraph $\mathcal{H}$.
Step 0: pre-partition $\mathbf{W}$ into p pieces (one-time cost)
Step 1: pick up partitions of $\mathbf{W}$ that contain all the query nodes to construct the new weighted graph $\mathbf{nW}$
Step 2:. run *CePS* as in table 3.1 on $\mathbf{nW}$

3.6 Experimental Evaluation

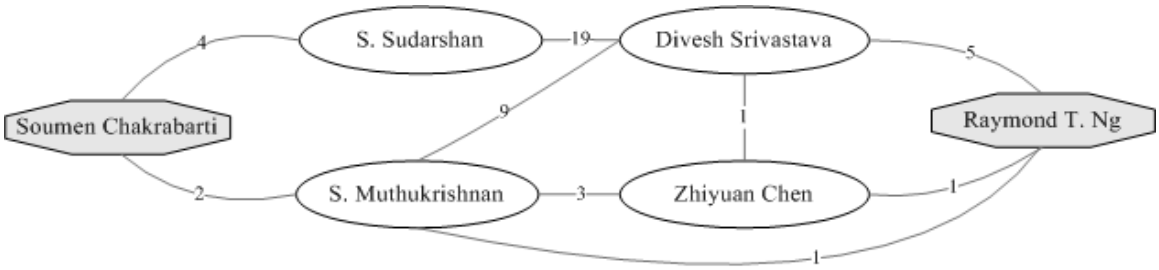
In this section, we demonstrate some experimental results. The experiments are designed to answer the following questions.

- Does the proposed goodness criterion make sense?
- Does the *EXTRACT* algorithm capture the most goodness score?
- Does the extra normalization step really help?
- how does the pre-partition balance the quality and response time?

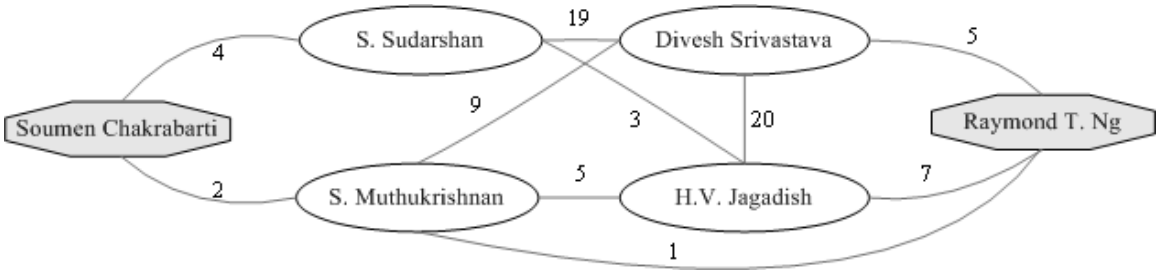
Data Set We use the DBLP data set to evaluate the proposed method. To be specific, the author-paper information is used to construct the weighted graph $\mathbf{W}$: every author is denoted as a



(a) by delivered current method ($+1$ voltage for Raymond and 0 voltage for Soumen)



(b) by delivered current method ($+1$ voltage for Soumen and 0 voltage for Raymond sink)



(c) by the proposed method

Figure 3.2: Connection subgraph between Soumen Chakrabarti and Raymond T. Ng.

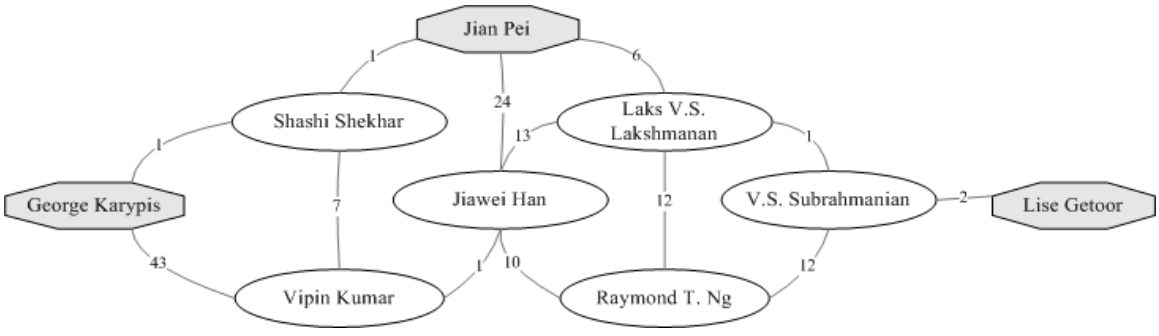


Figure 3.3: Center-piece subgraph among Lise Getoor, George Karypis, and Jian Pei.

node in $\mathbf{W}$; and the edge weight is the number of co-authored papers between the corresponding two authors. On the whole, there is $\approx 315K$ nodes and $\approx 1,834K$ non-zero edges in $\mathbf{W}$.

Source Queries To test the proposed algorithm, we select several people from different communities to compose the source-query repository: 13 people from database and mining; 13 people from statistical and machine learning; 11 people from information retrieval; and 11 people from computer vision. Then the source queries are generated by randomly selecting a small number of queries from the repository.

Parameter Setting The re-starting coefficient c in Eq. 3.4 is set 0.5 and the iteration number m is set 50 since we do not observe performance improvement with more iteration steps. The maximum allowable path length len is decided by the budget b and the number of active sources k as $[b/k]$. For normalization coefficient α, a parametric study is provided in Section 7.3. For other experiments, $\alpha = 0.5$.

Evaluation Criterion Firstly, the resulting $g(\mathcal{H})$ can be evaluated by "Important Node Ratio ($NRatio$)". That is, "how many important/good nodes are captured by $g(\mathcal{H})$?":

$$NRatio = \frac{\sum_{j\in\mathcal{H}} r(\mathcal{Q}, j)}{\sum_{j\in\mathbf{W}} r(\mathcal{Q}, j)} \tag{3.13}$$

Complementally, we can also evaluate by "Important Edge Ratio ($ERatio$)". That is, "how many important/good edges are captured by $g(\mathcal{H})$?":

$$ERatio = \frac{\sum_{(j,l)\in\mathcal{H}} r(\mathcal{Q}, (j,l))}{\sum_{(j,l)\in\mathbf{W}} r(\mathcal{Q}, (j,l))} \tag{3.14}$$

The goodness score $r(\mathcal{Q}, (j,l))$ of an edge (j,l) is defined similarly as the goodness score for a node: what is the probability that the specific edge (j,l) will be traversed simultaneously by all (or at least k) of the particles. Firstly, we calculate the goodness score $r(i, (j,l))$ for an edge (j,l) wrt a single query node q_i:

$$r(i, (j,l)) = \frac{1}{2} \cdot (r(i,j) \cdot \tilde{\mathbf{W}}_{l,j} + r(i,l) \cdot \tilde{\mathbf{W}}_{j,l}) \tag{3.15}$$

Based on Eq. 3.15, we can easily define $r(\mathcal{Q}, (j,l))$ according to the specific query type. For example, for "AND query", $r(\mathcal{Q}, (j,l))$ can be computed as Eq. 3.16; while for "OR query" and "K_softAND query", $r(\mathcal{Q}, (j,l))$ can be computed as Eq. 3.17 and Eq. 3.18, respectively.

$$r(\mathcal{Q}, (j,l)) \triangleq r(\mathcal{Q}, (j,l), Q) = \prod_{q_i=1}^{Q} r(i, (j,l)) \tag{3.16}$$

$$r(\mathcal{Q}, (j,l)) \triangleq r(\mathcal{Q}, (j,l), 1) = 1 - \prod_{q_i=1}^{Q} (1 - r(i, (j,l))) \tag{3.17}$$

$$
\begin{aligned}
r(\mathcal{Q},(j,l)) &\triangleq r(\mathcal{Q},(j,l),k) \\
&= r(\acute{\mathcal{Q}},(j,l),k-1)\cdot r(Q,(j,l)) + r(\acute{\mathcal{Q}},(j,l),k)
\end{aligned}
\tag{3.18}
$$

where $r(\varnothing,(j,l),0)=1$.

For all experiments except subsection 7.1, we run the proposed algorithm multiple times and report the mean $NRatio$ as well as mean $ERatio$.

3.6.1 Evaluation on the goodness $g(\mathcal{H})$: case study

As we mentioned before, connection subgraph is a special case of center-piece subgraph ("AND query" with pair source nodes). Figure 6.1 shows the connection subgraph with budget 4 for "Soumen Chakrabarti" and "Raymond T. Ng". It can be seen that both our method and the delivered current method output somewhat reasonable results. It is worth pointing out that the subgraph by the delivered current method is very sensitive to the order of the source queries: comparing figure 6.1(a) and (b), there is only one common node ("S. Muthukrishnan"). On the other hand, if we compare figure 6.1(b) and (c), while most nodes are the same for the two methods, It is clear that our method captures more strong connection: compared with figure 6.1(b), the different node ("H.V. Jagadish") in figure 6.1(c), 1) has more connections (4 vs. 3) with the remaining nodes and 2) has more co-authored papers with those connected neighbors than the corresponding node in figure 6.1(b) ("Zhiyuan Chen").

Figure 3.1 shows an example for multi-source queries. When the user asks for $2-SoftAND$, the algorithm outputs two clear cliques (figure 3.1(a)), which makes some sense since "Vladimir Vapnik" and "Michael I. Jordan" belong to statistical machine learning community; while "Rakesh Agrawal" and "Jiawei Han" are database and mining people. On the other hand, if the user asks for "AND", the resulting subgraph shows a strong connection with all four queries.

Figure 3.3 shows an example for "AND query", with "George Karypis", "Lise Getoor" and "Jian Pei" as source nodes. All three researchers are working on graphs. The nodes of the retrieved "center-piece subgraph" are all database, data mining and graph mining people, forming three groups: the nodes close to "Lise Getoor" are related to the University of Maryland ("V.S. Subrahmanian" is a faculty member there and he was the advisor of "Raymond Ng"). The nodes close to "George Karypis" are faculty members at Minnesota ("Vipin Kumar", "Shashi Shekar"). The nodes close to "Jian Pei" are professors at Simon Fraser (SFU) or University of British Columbia (UBC), which are geographically nearby, both in Vancouver: "Jiawei Han" was a faculty member at SFU and thesis advisor of "Jian Pei" ; "Laks Lakshmanan" and "Raymond Ng" are faculty members at UBC. Not surprisingly, the "center-pieces" of the subgraph consist of "Raymond Ng", "Jiawei Han", "Laks Lakshmanan", which all have direct, or strong indirect connections with the three chosen query sources.

3.6.2 Evaluation on "*EXTRACT*" algorithm

By the "*EXTRACT*" algorithm, we might miss some good/close nodes (which have high goodness scores) in order to meet the requirement of "connection". To evaluate this potential risk, we use both $NRatio$ and $ERatio$ as functions of the budget b (Higher $NRatio$ and $ERatio$ indicate lower risk). Here, we fix the query type as "AND query".

Figure 3.4(a) shows the mean $NRatio$ vs. the budget b for different numbers of source queries; while figure 3.4(b) shows the mean $ERatio$ vs. the budget b for different numbers of source queries. Note that in both cases, our method captures most of important nodes as well as edges by a small number of budget b. For example, for 2 source queries, the resulting subgraph with budget 50 captures 95% important nodes and 70% important edges on average; for 4 source queries, the resulting subgraph with budget 20 captures 100% important nodes and 70% important edges on average. An interesting observation is that for the same budget, the subgraph with more source queries captures higher $NRatio$ as well as $ERatio$ than those with less source queries. This is consistent with the intuition: generally speaking, finding people that are important wrt all source queries becomes more difficult when the number of source queries increases. In other words, $r(\mathcal{Q}, j)$ becomes more skewed by increasing the number of source queries.

3.6.3 Evaluation on normalization step

Here we conduct the parametric study for normalization coefficient α. The mean $NRatio$ vs. α is plotted in figure 3.5(a); and the mean $iERatio$ vs. α is plotted in figure 3.5(b).

It can be seen that in most cases, the normalization step does help to improve the performance of the resulting subgraph $g(\mathcal{H})$. For example, the normalization with $\alpha = 0.5$ helps to capture 17.7% more important nodes and 9.1% more important edges for 2 source queries on average; while for 3 source queries, it captures 18.1% more important nodes and 7.6% more important edges on average.

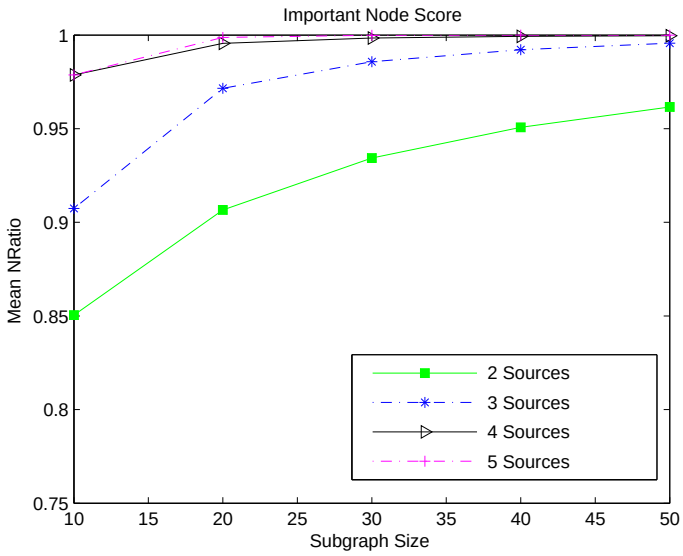
3.6.4 Evaluation on speedup strategy

For large graph, the response time for importance score calculation could be long. By pre-partition the original graph and performing subgraph discovery only on the partitions containing the source queries, we could dramatically reduce the response time. On the other hand, we might miss a few important nodes if they do not lie in these partitions. To measure such kind of quality loss, we use "Relative Important Node Ratio ($RelRatio$)":

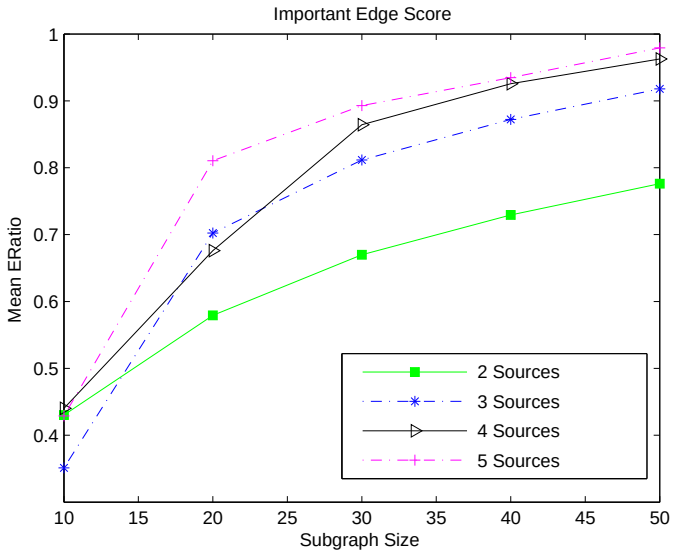
$$RelRatio = \frac{\widehat{NRatio}}{NRatio} \tag{3.19}$$

where $\widehat{NRatio}$ and $NRatio$ are "Important Node Ratio" for the subgraph by pre-partition and by the original whole graph, respectively.

We fix the budget 20 and the query scenario as "AND query". The mean $RelRatio$ vs. response time is shown in figure 3.6(a); and the mean response time vs. the number of partitions is shown in figure 3.6(b). It can be seen that with a little quality loss, the response process is largely speeded



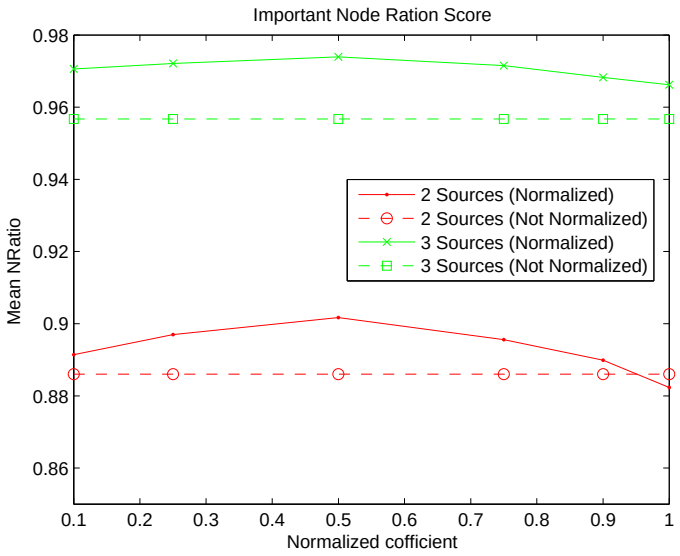
(a) Important node ratio vs. budget



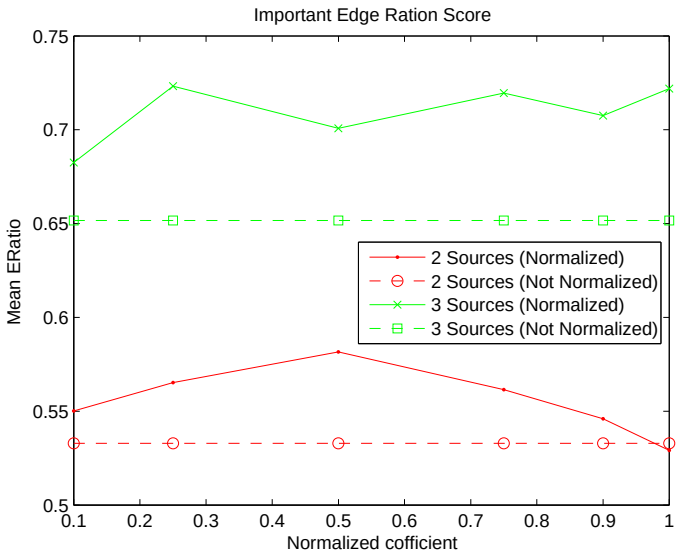
(b) Important edge ratio vs. budget

Figure 3.4: Evaluation on "*EXTRACT*". The proposed *CePS* captures most of important node/edge scores.

up. For example, with $\approx 10\%$ loss, the subgraph for 2 source queries can be generated within 5 seconds on average; with $\approx 10\%$ quality loss, the subgraph for 5 source queries can be generated within 10 seconds on average. On the other hand, it might take $40s \sim 60s$ without pre-partition. Note that in figure 3.6 (b), even with a small number of partitions, we can greatly reduce the mean response time.



(a) Important node ratio vs. α



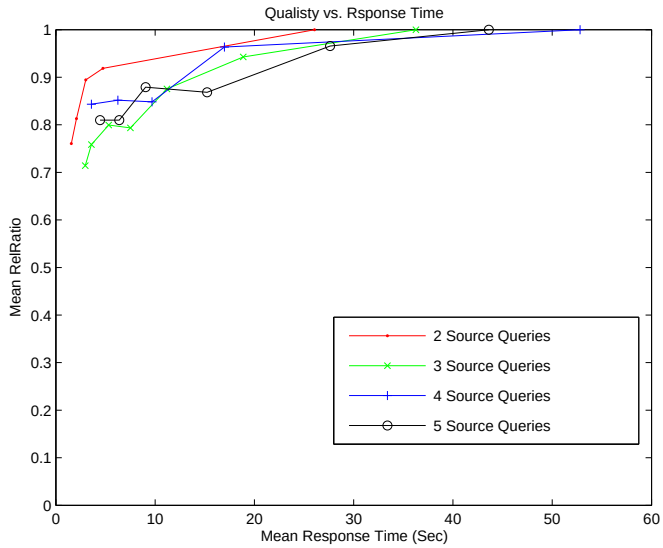
(b) Important edge ratio vs. α

Figure 3.5: Evaluation on normalization step

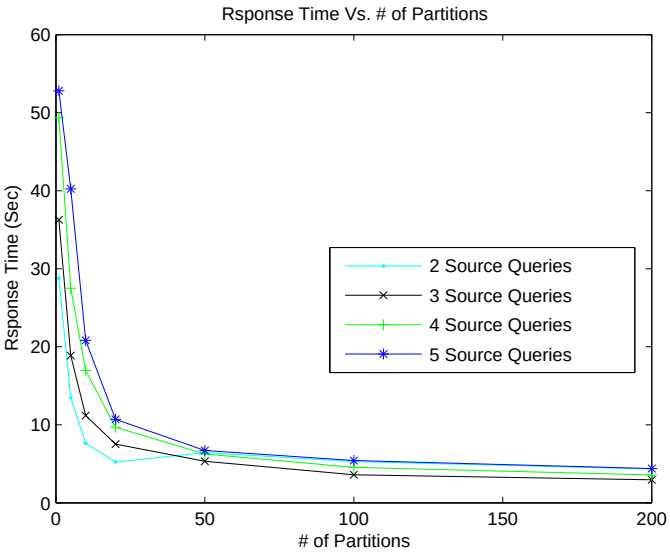
3.7 Related Work

Here, we make a brief review of the related work, which can be categorized into three groups: (1) measuring the goodness of closeness; (2) measuring the goodness of connection; (3) community mining; (4) random walk and electricity-based methods. The proposed *CePS* is also related to graph partition. For these work, please refer to Chapter 2.

Measuring the goodness of closeness. Defining a good closeness score is the core for center-



(a) Quality vs Time



(b) Time vs Number of partitions

Figure 3.6: Evaluation on speeding up strategy. The proposed Fast-*CePS* achieves about 10x speedup, with 90% quality preserving.

piece subgraph discovery. Here, the goal is to define a score to measure the closeness of a given node wrt the query set. To this end, we need to define a score to measure the closeness of a given node wrt a single query node. The two most natural measures for such purpose (i.e., the closeness between two nodes) are shortest distance and maximum flow. However, as pointed out in [FMT04], both measurements might fail to capture some preferred characteristics for social network. To be specific, shortest path will suffer from high degree nodes, and also it cannot capture the multiple

faceted relationship between two nodes on the graph; while maximum netflow does not punish the longer connections. The closeness function for survivable network [GMS93], which is the count of edge-disjoint or vertex-disjoint paths from source to destination, also fails to adequately model social relationship. A more related distance function is proposed in [LNK03] [PF03]. However, It cannot describe the multi-faceted relationship in social network since center-piece subgraph aims to discover collection of paths rather than a single path.

Measuring the goodness of connection. Another requirement in *CePS* is "connection". In [FMT04], the authors propose an delivered current based method. By interpreting the graph as an electric network, applying $+1$ voltage to one query node and setting the other query node 0 voltage, their method proposes to choose the subgraph which delivers maximum current between the query nodes. In [RMPS05], the authors further apply the delivered current based method to multi-relational graph. However, the delivered current criterion can only deal with pairwise source queries. Moreover, the resulting subgraph might be sensitive to the order of the query nodes (See Figure 6.1 for an example). On the other hand, as we will show very soon, connection subgraph can actually be viewed as a special case of the proposed center-piece subgraph ("AND query" with pair source nodes).

The "connection" requirement is also related to Steiner tree [CLR90, LTL03], where the goal is to find a tree of minimal weight which includes all query nodes. However, the Steiner tree cannot directly apply in our settings for the following reasons: (1) the Steiner tree might suffer from those high degree nodes exactly as the way the shortest path will suffer; (2) to find an exact Steiner tree is NP-complete; and (3) Steiner tree requires to find a tree which connects to all the source nodes. On the other hand, *CePS* tries to find a set of inter-correlated trees to connect the query nodes in an approximate way. By using the proposed closeness function, *CePS* will avoid the high-degree node effect. Also, in the proposed "EXTRACT" algorithm (which will be introduced Section 5), we try to search for a set of paths, instead of searching for a tree directly (as in Steiner tree). Finally, by introducing $K\_softAND$, we can further relax the requirement on connecting to all the source nodes in *CePS*.

Community detection. Center-piece subgraph discovery is also related with community detection, such as [FLGC02][GKR98][GN]. However, we cannot directly apply community detection to subgraph discovery especially when the source queries are remotely related or they lie in different communities.

Random walk related methods. The proposed importance score calculation is based on random walk with restart. There are many applications using random walk and related methods (See Chapter 2 for details).*CePS* also relates Personized PageRank (PPR) [FR04] in the sense that PPR defines the combined score as an approximate "OR " query[2]. On the other hand, the proposed *CePS* can naturally deal with different kinds of queries, from "AND " to "OR ", with "$K\_softAND$ query" in-between.

[2]To see this, notice that the combined score is defined as $r(\mathcal{Q}, j) = \sum_{i=1}^{Q} r(i, j)$ in PPR.

3.8 Conclusion and Discussions

Summary of Current Work. We have proposed the problem of "*center-piece subgraphs*", and provided fast and effective solutions. In addition to the problem definition, other contributions of the chapter are the following:

- The introduction and handling of $K\_softAND$ queries, which include AND and OR queries as special cases.
- *EXTRACT*, a fast novel algorithm to quickly extract a subgraph with the appropriate connectivity and maximum "goodness" score
- The design and implementation of a fast, approximate algorithm that brings a 6:1 speedup
- Experiments on real data (DBLP), illustrating that our algorithm and "goodness score" indeed derive results that agree with intuition.

Discussions and Future Work. In this chapter, we have focused on the plain graph (i.e., no attributes on the nodes or edges). And also, we have restricted to the un-directed graphs. In [TKF07], we have generalized *CePS* to the directed graphs. And in [TFGER07], we have generalized *CePS* to the attributed graphs.

In the future, we would also like to investigate this problem in the following aspects:

1. Automatic parameter tuning. For example, if the user does not provide the $K\_softAND$ coefficient, how can we infer the 'optimal' k. One possible way to attach this problem is through cross validation (by treating *CePS* as a retrieval/classification tool.
2. Steiner tree and *CePS*. For example, how to leverage the approximate algorithms for Steiner tree so that we can provide theoretic performance guarantee for *CePS*; how to generalize the Steiner tree by *CePS* (e.g., to find a set of inter-correlated, rather than one, Steiner tree; to find the "soft" Steiner tree which connects at least k-out-of-Q queries node etc).

Chapter 4

Case Study #2: User Feedback

Summary of This Chapter

- **Questions we want to answer:**
  - Q1: How to incorporate users like/dislike type of feedback in measuring proximity on graphs?
  - Q2: How to reflect users (near) real time interest?
- **Our answers and contributions**
  - A1: We proposed a novel method (iPoG) to incorporate user feedback (like/dislike) in measuring node proximity on large graphs, enriching a broad range of applications.
  - A2: We proposed a fast algorithm (Fast-iPoG) to compute the proposed proximity measurement, achieving significant speedups (up to 49x).

4.1 Introduction

Most existing work on querying static graphs only considers the link structure of the underlying graph, ignoring any possible side information. For example, given an author-conference bipartite graph, existing proximity measurements may answer the question: *What are the most similar conferences to KDD?* However, for a particular user, s/he might have her/his own preferences: *I dislike ICML or I like SIGIR*. These preferences are typically localized to a particular search, and may not reflect a global sentiment by the user.

There are a wide range of scenarios where users' feedback, both implicit or explicit, can be naturally integrated as side information.[1] For instance, in recommendation systems, side information could be users' ratings on items (e.g., *I like Kung-Fu Panda*). In Blog analysis, it could be opinions and sentiments. Additionally, for many real applications, users' preferences can be estimated from

[1] In this chapter, we use the terms 'user feedback' and 'side information' interchangeably.

click-through data. That said, it is thus important to incorporate such side information in the proximity measurement so that search results are well-tailored to reflect a user's individual preferences. In the earlier example, the question will then become: *What are the most similar conferences to KDD, but dissimilar to ICML?*

In this chapter, we address the above challenge by proposing a novel method, called iPoG, that incorporates such like/dislike side information in measuring node proximity on large graphs. Our method is based on random walk with restart (RWR), where iPoG uses the side information to refine the graph structure so that RWR is biased to avoid or to favor some specific zones on the graph according to the users' preferences. Additionally, iPoG inherits existing capabilities from RWR, such as the ability to summarize the *multiple faceted* relationships, to be interpreted from the perspective of *steady-state probability*, etc. Therefore, we expect iPoG to enrich a broad-range of applications by replacing their original proximity measurement implementation. We evaluate iPoG in three case studies: neighborhood search, center-piece subgraph, and image caption. In all cases, we show that iPoG naturally reflects the users' preference and/or improves the quality of the existing applications (e.g., boost the precision/recall of the image captions by more than 10%).

Because a straightforward implementation of iPoG requires significant computation, we propose a fast algorithm (Fast-iPoG) that computes the proposed proximity measurement, while radically reducing the computational overhead. Fast-iPoG achieves the performance gains by exploiting the smoothness of the graph structures with/without side information. Our experimental results show that it achieves significant speedup (*up to 49x*) while maintaining high approximation accuracy (more than 93.0%).

This chapter has three key contributions:

- A novel method (iPoG) to incorporate side information (like/dislike) in measuring node proximity on large graphs, enriching a broad range of applications;
- A fast algorithm (Fast-iPoG) to compute the proposed proximity measurement, achieving significant speedups (up to 49x);
- Extensive experimental evaluations on several real datasets.

The rest of this chapter is organized as follows. We introduce notations and formally define the problem in Section 4.2. We present the proposed proximity measurement in Section 4.3 and the fast algorithm in Section 4.4, respectively. We provide experimental evaluations in Section 4.5 and review the related work in Section 4.6. Finally, we conclude in Section 4.7.

4.2 Problem Definitions

Table 4.1 lists the main symbols that we use throughout this chapter. We represent a general graph by its adjacency matrix. Following the standard notation, we use capital letters for matrices (e.g. $\mathbf{A}$), lower case for vectors (e.g. $\mathbf{a}$), and calligraphic fonts for sets (e.g. $\mathcal{I}$). We use the symbol " $\tilde{}$ " to distinguish the setting with/without side information. For example, $\mathbf{A}$ is the normalized adjacency matrix of the graph without side information; and $\tilde{\mathbf{A}}$ is the normalized adjacency matrix of the refined graph by side information.

We represent the elements in a matrix using a convention similar to Matlab, e.g., $\mathbf{A}(i, j)$ is the

Table 4.1: Symbols

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}, \ldots$	matrices (bold upper case)
$\mathbf{A}(i,j)$	element at the i^{th} row and j^{th} column of $\mathbf{A}$
$\mathbf{A}(i,:)$	i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:,j)$	j^{th} column of matrix $\mathbf{A}$
$\mathbf{a}, \mathbf{b}, \ldots$	column vectors
$\mathcal{I}, \mathcal{J}, \ldots$	sets (calligraphic)
n	number of nodes in the graph
n^i	number of out links of node i
c	$(1-c)$ is the restart probability
$r_{i,j}$	proximity from node i to node j
$\mathbf{r}_i = [r_{i,j}]$	ranking vector for node i $(j = 1, ..., n)$
$\mathcal{P}$	positive set $\mathcal{P} = \{x_1, ..., x_{n^+}\}$
$\mathcal{N}$	negative set $\mathcal{N} = \{y_1, ..., y_{n^-}\}$
n^+	number of positive nodes $n^+ = \lvert\mathcal{P}\rvert$
n^-	number of negative nodes $n^- = \lvert\mathcal{N}\rvert$
$\mathbf{e}_i$	$n \times 1$ starting vector for node i, where $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0 (j \neq i)$

element at the i^{th} row and j^{th} column of the matrix $\mathbf{A}$, and $\mathbf{A}(:,j)$ is the j^{th} column of $\mathbf{A}$, etc.

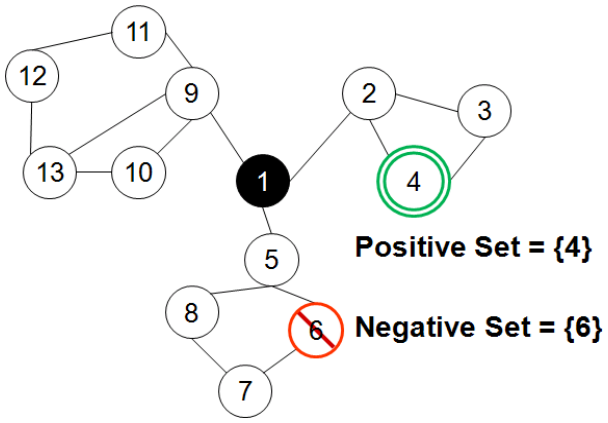
We use a running example, depicted in Fig.5.1(a), to describe the problem statement. There, each node represents a person (e.g., node 1 is 'John', node 2 is 'Smith', etc.) and the existence of edge represents some social contact between the two corresponding persons (e.g., phone call). In traditional settings of proximity measurement, the goal is to quantify the closeness (i.e., relevance) between two nodes (the source and target) based on the link structure of the underlying graph. In our settings, we assume the existence of side information, focusing primarily on like/dislike user feedback as side information. In our running example, a user might not want to see (i.e., dislike) node 6 but favors (i.e., like) node 4.

Formally, we represent such side information by two sets $\mathcal{P}$ and $\mathcal{N}$. The set $\mathcal{P}$ contains the node indices that users like (referred to as the positive set), where the corresponding nodes are referred as positive nodes. The set $\mathcal{N}$ contains the node indices that users dislike (referred as negative set), where the corresponding nodes are referred to as negative nodes. In our running example, both the positive set $\mathcal{P}$ and the negative set $\mathcal{N}$ contain one single element, respectively: $\mathcal{P} = \{4\}$ and $\mathcal{N} = \{6\}$. Our goal is to incorporate such side information to measure the node proximity (e.g., the proximity from node 1 to the node 3 in our running example).

With the above notations and assumptions in mind, our problem can be formally defined as follows:

Problem 2. (Proximity with Side Information)

Given: *a weighted direct graph* $\mathbf{A}$*, a source node* s *and a target node* t*, and side information* $\mathcal{P}$ *and* $\mathcal{N}$*;*



(a) the graph (node 1 is the source.)

$$\begin{pmatrix}
0 & 0.33 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 \\
0.33 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.33 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0 & 0.5 & 0
\end{pmatrix}$$

(b) column normalized adjacency matrix $\mathbf{A}$

Figure 4.1: The running example.

Find: *the proximity score $\tilde{r}_{s,t}$ from source node s to target node t.*

In problem 2, if the target node t is absent, we measure the proximity score $\tilde{r}_{s,i}(i = 1, ..., n)$ from the source node s to all the other nodes in the graph. If we stack all these scores into a column vector $\tilde{\mathbf{r}}_s = [\tilde{r}_{s,i}](i = 1, ..., n)$, it is equivalent to saying that we want to compute the ranking vector $\tilde{\mathbf{r}}_s$ for the source node s. In this chapter, we assume that there is no overlap between the positive set and negative set (i.e., $\mathcal{P} \cap \mathcal{N} = \phi$.[2]) Also, the positive and negative side information do not need to exist simultaneously. For example, if we only have positive side information, we can simply set the negative set to be empty (i.e., $\mathcal{N} = \phi$).

[2]If this does not hold, we can remove the intersection from both positive set and negative set.

4.3 iPoG

In this section, we introduce our proximity measurement with side information, iPoG. We begin by reviewing random walk with restart (RWR), which is a good proximity measurement for the case where there is no side information. We, then, extend RWR to properly account for side information.

4.3.1 RWR: Proximity without Side Information

Random walk with restart (RWR) is considered one of the most successful methods for measuring proximity and is receiving increased interest in recent years. For a given graph, RWR is defined as follows. Consider a random particle that starts from node i. The particle iteratively transits to its neighbors with probabilities proportional to the corresponding edge weights. At each step, the particle can 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 [PYFD04]. 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 stack all the proximity scores $r_{i,j}$ into a column $\mathbf{r}_i$ (referred to as the ranking vector for the node i), the equation (4.1) gives the formal definition of RWR:

$$\mathbf{r}_i = c\mathbf{A}\mathbf{r}_i + (1-c)\mathbf{e}_i, \tag{4.1}$$

where $\mathbf{A}$ is the column normalized adjacency matrix for the graph and $\mathbf{e}_i$ is the starting vector for node i.

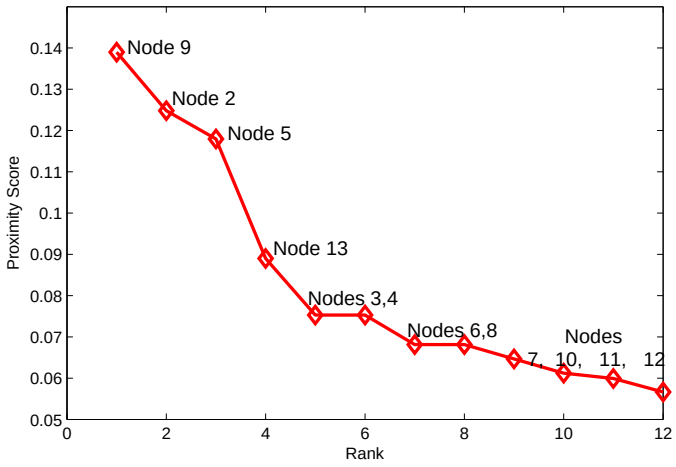
For our running example in Fig. 5.1(a), its normalized adjacency matrix $\mathbf{A}$ is shown in Fig. 5.1(b). If we ignore any side information, by setting the correct starting vector (e.g., $\mathbf{e}_1 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]'$ for node 1), we can solve the corresponding ranking vector using equation (4.1). Fig. 4.2(a) plots the ranking vector (sorted from highest to lowest) for node 1 of the running example. The scores are consistent with our intuition: nearby nodes (e.g., nodes 9, 2 and 5) receive higher proximity scores.

4.3.2 iPoG: Proximity with User Feedback

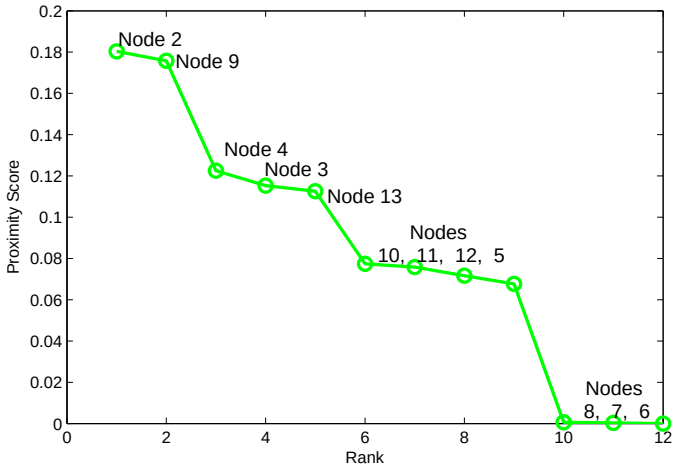
Basic Ideas. Our goal is to incorporate side information to measure the node proximity. Intuitively, for a given source node s, if positive nodes exist, the proximity score from the source node to such positive nodes as well as their neighboring nodes should increase, compared to the case where such side information is unavailable. In our running example, if we know node 4 belongs to the positive set $\mathcal{P}$, we expect that the proximity score from the source node 1 to node 4 to increase and so will the proximity scores from node 1 to node 4's neighboring nodes (e.g., node 2 and node 3). Analogously, if negative nodes exist, the proximity scores from the source node to such negative nodes as well as their neighboring nodes should decrease, compared to the case where such side information is unavailable. In our running example, if we know that node 6 belongs to the negative set $\mathcal{N}$, we expect the proximity score from node 1 to node 6 to decrease, and so will node 6's neighboring nodes (such as nodes 5 and 7).

The basic idea of iPoG is then to use side information to refine the original graph structure so that the random particle (1) has higher chances of visiting the positive nodes as well as their neighboring nodes, and (2) has lower chances of visiting the negative nodes as well as their neighboring nodes.

Dealing with Positive Nodes. For each node x in the positive set ($\mathcal{P}$), we create a direct link from the source node s to node x. As in the running example, we add a direct link from the source node 1 to node 4 (See Fig. 4.3(a)). In this way, whenever the random particle visits (or restarts from) the source s, it has higher chances of visiting the nodes in the positive set. Note that we are

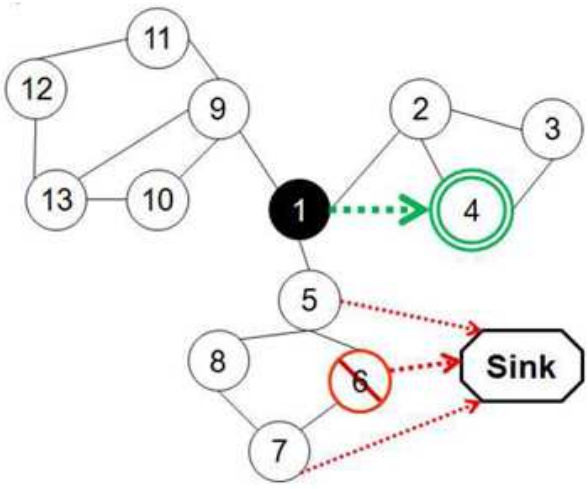


(a) without side information



(b) with side information

Figure 4.2: Ranking vector for node 1 in the running example in Fig. 5.1. (The proximity scores are normalized so that they sum up to 1.)



(a) the updated graph

$$\begin{pmatrix}
0 & 0.33 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 \\
0.25 & 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.33 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.25 & 0.33 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.01 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.01 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0.25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.33 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0.5 & 0 & 0.5 & 0
\end{pmatrix}$$

(b) updated column normalized adjacency matrix

Figure 4.3: Adjustment on the original graph in the running example in Fig. 5.1.

also implicitly increasing the chance that the random particle will visit the neighborhood of those positive nodes. The weight of each newly added link is set to $1/(n^s + n^+)$. For example, the newly added edge (1,4) for the running example will receive a weight of 0.25 (since $n^1 = 3$ and $n^+ = 1$).

Dealing with Negative Nodes. To deal with the negative nodes, we introduce a sink into the graph, which has no out links. For each node y in the negative set ($\mathcal{N}$), we put a direct link from node y to the sink. Thus, whenever the random particle visits this node, it can go to the sink and never comes back (since there is no out links from the sink). Therefore, this negative node y is penalized and its corresponding proximity score will decrease. In order to penalize the neighborhood of node y, we also put a direct link from its neighboring nodes to the sink. In our

running example, besides the link from node 6 (the negative node) to the sink, we placed a link from nodes 5 and 7 (the neighboring nodes of node 6) to the sink respectively (see Fig. 4.3(a)).

There are two remaining questions: (1) how to choose the neighborhood of a negative node y and (2) how to determine the weights to the sink. Let the index of the sink node be $n+1$, the procedure is summarized in Alg. 1. In Alg. 1, we use random walk with restart (on the original graph) to determine (1) the neighborhood of the negative node y (steps 2-4), and (2) the weights of the newly added links to the sink (steps 5-6). Notice that we eventually (step 9) discard the last row/column (which corresponds to the sink node). We use it to simplify the description of the proposed method without affecting the ranking vector in accord to the property of a sink node.

Algorithm 1 Add Links for One Negative Node

Require: The adjacency matrix $\mathbf{A}$, the negative node y, the neighborhood size k and c.
Ensure: The updated adjacency matrix $\tilde{\mathbf{A}}$.

1: initialize $\tilde{\mathbf{A}} = \mathbf{A}$, $\tilde{\mathbf{A}}(n+1,:) = 0$, and $\tilde{\mathbf{A}}(:,n+1) = 0$.
2: get the ranking vector for the negative node y by $\mathbf{r}_y = c\mathbf{A}\mathbf{r}_y + (1-c)\mathbf{e}_y$. Let $\epsilon := k^{\text{th}}$ largest element in $\mathbf{r}_y$.
3: **for** each node i **do**
4: **if** $\mathbf{r}_{y,l} \geq \epsilon$ **then**
5: set $\tilde{\mathbf{A}}(n+1,l) = \mathbf{r}_{y,i}/\mathbf{r}_{y,y}$
6: set $\tilde{\mathbf{A}}(1:n,i) = (1-\mathbf{r}_{y,i}/\mathbf{r}_{y,y})\tilde{\mathbf{A}}(1:n,i)$
7: **end if**
8: **end for**
9: output $\tilde{\mathbf{A}} = \tilde{\mathbf{A}}(1:n,1:n)$.

iPoG Algorithm. Based on the above preparations, the complete algorithm to measure proximity with side information (iPoG) is given in Alg. 2. In Alg. 2, after initialization (step 1), we first use side information to refine the graph structure (steps 2-7 for positive nodes,[3] and steps 8-12 for negative nodes). Note that in step 10, we use the same $\mathbf{A}$ (i.e., the original graph) to add links for each negative node y. This is because we assume that all the negative nodes are obtained in a batch mode (i.e., there is no ordering among different negative nodes). Then, we perform a random walk with restart on the refined graph ($\tilde{\mathbf{A}}$) for the source node s (step 13) and output the corresponding steady state probability as the proximity score (step 14). For example, Fig. 4.2(b) plots the ranking vector (sorted from highest to lowest) for node 1 of the running example with side inforamtion ($\mathcal{P} = \{4\}$, and $\mathcal{N} = \{6\}$). Compared to the case without side information (Fig. 4.2(a)), it can be seen that positive node (node 4) as well as its neighborhood (nodes 2 and 3) receives higher proximity scores; while the negative node (node 6) as well as its neighboring nodes (nodes 5 and 7) receives lowers scores.

[3]Note that step 3 is to insure that the s^{th} column of $\tilde{\mathbf{A}}$ sums up to 1.

Algorithm 2 iPoG

Require: The adjacency matrix $\mathbf{A}$, the source node s and the target node t, the side information $\mathcal{P}$ and $\mathcal{N}$, the neighborhood size k, and the parameter c.

Ensure: the proximity score $\tilde{\mathbf{r}}_{s,t}$ from source s to target t.

1: initialize $\tilde{\mathbf{A}} = \mathbf{A}$
2: **if** $n^+ > 0$ **then**
3: $\quad \tilde{\mathbf{A}}(:, s) = n^s/(n^s + n^+)\tilde{\mathbf{A}}(:, s)$
4: $\quad$ **for** each positive node x in $\mathcal{P}$ **do**
5: $\quad\quad \tilde{\mathbf{A}}(x, s) = \tilde{\mathbf{A}}(x, s) + 1/(n^s + n^+)$.
6: $\quad$ **end for**
7: **end if**
8: **if** $n^- > 0$ **then**
9: $\quad$ **for** each negative node y in $\mathcal{N}$ **do**
10: $\quad\quad$ update $\tilde{\mathbf{A}}$ by Alg. 1
11: $\quad$ **end for**
12: **end if**
13: solve the equation $\tilde{\mathbf{r}}_s = c\tilde{\mathbf{A}}\tilde{\mathbf{r}}_s + (1 - c)\mathbf{e}_s$.
14: output $\tilde{\mathbf{r}}_{s,t} = \tilde{\mathbf{r}}_s(t)$.

4.4 Fast-iPoG

In this section, we introduce our fast solution for iPoG. We start by reviewing NB_LIN, which is a fast algorithm to compute random walk with restart (the proximity without side information. See Chapter 2). We then extend it to include side information.

4.4.1 Background: NB_LIN for RWR

According to the definition (equation (4.1)), we need to invert an $n \times n$ matrix. This operation is prohibitively slow for large graphs. On the other hand, the iterative method (iterating equation (4.1) until convergence) might need many iterations, which is also not efficient. In [TFP08], the authors solve this problem using a low-rank approximation, followed by a matrix inversion of size $l \times l$ (where l is the rank of the low-rank approximation) to get all possible proximity scores. Their solution, called NB_LIN, is the starting point for our fast algorithm.

Alg. 3 summarizes NB_LIN, where it is divided into two stages: $\mathrm{NB\_LIN\_Pre}()$ and $\mathrm{NB\_LIN\_OQ}()$. In $\mathrm{NB\_LIN\_Pre}()$ (steps 1-3), a low-rank approximation is performed for the normalized adjacency matrix $\mathbf{A}$ and a matrix inversion $\mathbf{\Lambda}$ is computed. Next, in $\mathrm{NB\_LIN\_OQ}()$ (steps 4-5), only a small number of matrix-vector multiplications are computed to output the ranking vector.

4.4.2 Fast-iPoG

To incorporate side information, we need to solve random walk with restart in two places. First, we process the original graph $\mathbf{A}$ (step 10 in Alg. 4); and then we process the refined graph $\tilde{\mathbf{A}}$ to get the

Algorithm 3 NB_LIN

Require: The normalized adjacency matrix $\mathbf{A}$, the source node s and c.
Ensure: The ranking vector for source node $\mathbf{r}_s$.
1: **Pre-Compute Stage** ($\mathrm{NB\_LIN\_Pre}()$)
2: do low-rank approximation for $\mathbf{A} = \mathbf{USV}$
3: pre-compute and store the matrix $\mathbf{\Lambda} = (\mathbf{S}^{-1} - c\mathbf{VU})^{-1}$
4: **On-Line Query Stage** ($\mathrm{NB\_LIN\_OQ}()$)
5: output $\mathbf{r}_s = (1-c)(\mathbf{e}_s + c\mathbf{U\Lambda V e}_s)$

ranking vector for the source node s (step 13 in Alg. 4). If we utilize NB_LIN in a straightforward way, we have to call it twice (for $\mathbf{A}$ and for $\tilde{\mathbf{A}}$, respectively). Unfortunately, this does not fit the expect usage model of side information, where it needs to reflect users' real-time interests. Imagine a user is querying an author-conference bipartite graph, and s/he wants to know *which conferences are most similar to KDD*. After the system gives the initial search results, s/he might further give her/his own preference (e.g., *dislike ICML*) and expect updated search results that matches her/his interests. This basically implies that calling $\mathrm{NB\_LIN\_Pre}()$ on the refined graph $\tilde{\mathbf{A}}$ is part of the on-line cost, which may pose a huge threat to the system's performance.

To deal with such challenge, we propose Fast-iPoG, which is given in Alg. 4. Here, we assume that we want the whole ranking vector for a given source node s since a single proximity score can be read out from such ranking vector. Also, we consider the most general case, where both positive nodes and negative nodes are present. In Fast-iPoG, it first calls $\mathrm{NB\_LIN\_Pre}()$ on the original adjacency matrix $\mathbf{A}$ (step 2). Then it calls $\mathrm{NB\_LIN\_OQ}()$ to determine the influence of the negative nodes (steps 5-12) and partial influence (i.e., scaling the s^{th} column of the adjacency matrix by a factor of $n^s/(n^s+n^+)$) of positive nodes (step 13), both of which are used to update the low-rank approximation ($\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$) as well as matrix $\tilde{\mathbf{\Lambda}}$ (steps 14 - 21). This way, it avoids directly calling the function $\mathrm{NB\_LIN\_Pre}()$ on the refined graph $\tilde{\mathbf{A}}$, where it would need to do a low-rank approximation and a matrix inversion, both of which are not efficient as on-line costs. Finally, it calls $\mathrm{NB\_LIN\_OQ}()$ twice (steps 23-24) and combines them as the final ranking result (step 25). Note that the second call on $\mathbf{e}_+$ (step 24) is used to compensate for the remaining influence of the positive nodes (i.e., adding new links from the source to the positive nodes).

The correctness of Alg. 4 is guaranteed by theorem 1. By theorem 1, Fast-iPoG will not introduce additional approximation errors beyond the first time it calls $\mathrm{NB\_LIN\_Pre}()$ on the original graph. Therefore, Fast-iPoG is expected to obtain ranking results similar to calling $\mathrm{NB\_LIN\_Pre}()$ twice (one for $\mathbf{A}$ and the other for $\tilde{\mathbf{A}}$). On the other hand, Fast-iPoG avoids the expensive steps (low-rank approximation on $\tilde{\mathbf{A}}$ and a matrix inversion of size $l \times l$) in calling $\mathrm{NB\_LIN\_Pre}()$. This, as we will show, leads to significant on-line running cost savings.

Theorem 1. Correctness of Fast-iPoG. *If* $\mathbf{A} = \mathbf{USV}$ *holds, then Alg. 4 gives the correct ranking vector for the source node* s.

Proof: let an $n \times n$ matrix $\hat{\mathbf{A}}$ s.t.,

$$\begin{aligned} \hat{\mathbf{A}}(:, \mathbf{\Theta}(j,1)) &= \mathbf{A}(:, \mathbf{\Theta}(j,1))\mathbf{\Theta}(j,1) \ (j = 1 : kn^- + 1) \\ \hat{\mathbf{A}}(:,i) &= \mathbf{A}(:,i) \quad \text{if} \quad i \notin \mathbf{\Theta}(:,1) \end{aligned} \tag{4.2}$$

Algorithm 4 Fast-iPoG

Require: The adjacency matrix $\mathbf{A}$, the source node s, the side information $\mathcal{P}$ and $\mathcal{N}$, the neighborhood size k, and the parameter c.

Ensure: the ranking vector $\tilde{\mathbf{r}}_s$ for the source s.

1: **Pre-Compute Stage**
2: call $[\mathbf{U}, \mathbf{\Lambda}, \mathbf{V}] = \text{NB\_LIN\_Pre}(\mathbf{A}, c)$
3: **On-Line Query (Feedback) Stage**
4: initialize $i_0 = 1$ and $\mathbf{\Theta} = \mathbf{0}^{(kn^- + 1) \times 2}$
5: **for** each negative node y in $\mathcal{N}$ **do**
6: call $\mathbf{r}_y = \text{NB\_LIN\_OQ}(c, \mathbf{U}, \mathbf{\Lambda}, \mathbf{V}, \mathbf{e}_y)$.
7: let $\epsilon := k^{\text{th}}$ largest element in $\mathbf{r}_y$.
8: **for** each node i s.t. $\mathbf{r}_{y,i} >= \epsilon$ **do**
9: set $\mathbf{\Theta}(i_0, 1) = i$ and $\mathbf{\Theta}(i_0, 2) = 1 - \mathbf{r}_{y,i}/\mathbf{r}_{y,y}$
10: increase i_0 by 1
11: **end for**
12: **end for**
13: set $\mathbf{\Theta}(i_0, 1) = s$ and $\mathbf{\Theta}(i_0, 2) = n^s/(n^s + n^+)$
14: set $\tilde{\mathbf{U}} = \mathbf{U}$ and $\tilde{\mathbf{V}} = \mathbf{V}$
15: **for** $i = 1 : kn^- + 1$ **do**
16: set $\mathbf{X}(i, :) = \mathbf{U}(\mathbf{\Theta}(i, 1), :)$
17: set $\mathbf{Y}(:, i) = \mathbf{V}(:, \mathbf{\Theta}(i, 1))(\mathbf{\Theta}(i, 2) - 1)$
18: set $\mathbf{V}(:, \mathbf{\Theta}(i, 1)) = \mathbf{V}(:, \mathbf{\Theta}(i, 1))\mathbf{\Theta}(i, 2)$
19: **end for**
20: compute $\mathbf{L} = (\mathbf{I} - c\mathbf{X}\mathbf{\Lambda}\mathbf{Y})^{-1}$
21: update $\tilde{\mathbf{\Lambda}} = \mathbf{\Lambda} + c\mathbf{\Lambda}\mathbf{Y}\mathbf{L}\mathbf{X}\mathbf{\Lambda}$
22: set $\mathbf{e}_+ = \mathbf{0}^{n \times 1}$, $\mathbf{e}_+(\mathcal{P}) = 1/(n^s + n^+)$
23: call $\hat{\mathbf{r}}_s = \text{NB\_LIN\_OQ}(c, \tilde{\mathbf{U}}, \tilde{\mathbf{\Lambda}}, \tilde{\mathbf{V}}, \mathbf{e}_s)$
24: call $\mathbf{u} = \text{NB\_LIN\_OQ}(c, \tilde{\mathbf{U}}, \tilde{\mathbf{\Lambda}}, \tilde{\mathbf{V}}, \mathbf{e}_+)$
25: output $\tilde{\mathbf{r}}_s = \hat{\mathbf{r}}_s + c\hat{\mathbf{r}}_s(s)/(1 - c - c\mathbf{u}(s))\mathbf{u}$

First, we will show that $\hat{\mathbf{r}}_s$ in step 23 gives the correct ranking vector on the matrix $\hat{\mathbf{A}}$ if $\mathbf{A} = \mathbf{USV}$ holds.

By the construction of matrix $\hat{\mathbf{A}}$, we have

$$\begin{aligned} \hat{\mathbf{A}}(:, \mathbf{\Theta}(j, 1)) &= \mathbf{USV}(:, \mathbf{\Theta}(j, 1))\mathbf{\Theta}(j, 1) \ (j = 1 : kn^- + 1) \\ \hat{\mathbf{A}}(:, i) &= \mathbf{USV}(:, i) \quad \text{if} \quad i \notin \mathbf{\Theta}(:, 1) \end{aligned} \tag{4.3}$$

Thus, in the matrix form, we have $\hat{\mathbf{A}} = \tilde{\mathbf{U}}\mathbf{S}\tilde{\mathbf{V}}$, where the matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are as defined in steps 14-19 in Alg. 4.

Define the matrix $\hat{\mathbf{Q}} = (1 - c)(\mathbf{I} - c\hat{\mathbf{A}})^{-1}$. By the property of NB_LIN algorithm [TFP08], we

have

$$\begin{aligned}\hat{\mathbf{Q}} &= (1-c)(\mathbf{I}-c\hat{\mathbf{A}})^{-1} \\ &= (1-c)(\mathbf{I}-c\tilde{\mathbf{U}}\mathbf{S}\tilde{\mathbf{V}})^{-1} \\ &= (1-c)(\mathbf{I}+c\tilde{\mathbf{U}}\hat{\mathbf{\Lambda}}\tilde{\mathbf{V}}) \end{aligned} \tag{4.4}$$

where $\hat{\mathbf{\Lambda}} = (\mathbf{S}^{-1} - c\tilde{\mathbf{V}}\tilde{\mathbf{U}})^{-1}$.

Next, we will relate $\hat{\mathbf{\Lambda}}$ with the matrix $\tilde{\mathbf{\Lambda}}$ (step 21 of Alg. 4).

By the spectral representation, we have the following equation:

$$\begin{aligned}\mathbf{S}^{-1} - c\tilde{\mathbf{V}}\tilde{\mathbf{U}} &= \mathbf{S}^{-1} - c\sum_i \tilde{\mathbf{V}}(:,i)\tilde{\mathbf{U}}(i,:) \\ &= \mathbf{S}^{-1} - c(\sum_i \mathbf{V}(:,i)\mathbf{U}(i,:) + \delta) \end{aligned} \tag{4.5}$$

where δ satisfies

$$\begin{aligned}\delta &= \sum_{j=1}^{kn^-+1} \mathbf{V}(:,\mathbf{\Theta}(j,1))\mathbf{U}(\mathbf{\Theta}(j,1),:)(\mathbf{\Theta}(j,2)-1) \\ &= \mathbf{YX} \end{aligned} \tag{4.6}$$

where the matrices $\mathbf{Y}$ and $\mathbf{X}$ are defined as steps 16-17 of Alg. 4.

Plugging equations (6.5) and (6.6) into the matrix $\hat{\mathbf{\Lambda}}$ and applying Sherman-Morrison Lemma [PC90], we have

$$\begin{aligned}\hat{\mathbf{\Lambda}} &= (\mathbf{S}^{-1} - c\tilde{\mathbf{V}}\tilde{\mathbf{U}})^{-1} \\ &= \mathbf{\Lambda} + c\mathbf{\Lambda}\mathbf{Y}\mathbf{L}\mathbf{X}\mathbf{\Lambda} \\ &= \tilde{\mathbf{\Lambda}} \end{aligned} \tag{4.7}$$

where the matrices $\tilde{\mathbf{\Lambda}}$ and $\mathbf{L}$ are defined as steps 20-21 of Alg. 4.

Plugging equation (6.6) into equation (4.4), we can verify the $\hat{\mathbf{r}}_s$ in step 23 satisfies:

$$\hat{\mathbf{r}}_s = \hat{\mathbf{Q}}(:,s) \tag{4.8}$$

Next, define the matrix $\tilde{\mathbf{Q}} = (1-c)(\mathbf{I}-c\tilde{\mathbf{A}})^{-1})$. We will try to relate $\tilde{\mathbf{Q}}$ with matrix $\hat{\mathbf{Q}}$.

By the construction of $\tilde{\mathbf{A}}$ and $\hat{\mathbf{A}}$, we have

$$\tilde{\mathbf{A}} = \hat{\mathbf{A}} + \mathbf{e}_+\mathbf{e}_s' \tag{4.9}$$

where vector $\mathbf{e}_+$ is defined as in step 22. In other words, there is only a rank-1 difference between $\tilde{\mathbf{A}}$ and $\hat{\mathbf{A}}$.

Now, applying Sherman-Morrison Lemma [PC90] to $\tilde{\mathbf{Q}}$, we have

$$\begin{aligned}\tilde{\mathbf{Q}} &= (1-c)(\mathbf{I}-c\tilde{\mathbf{A}})^{-1}) \\ &= (1-c)(\mathbf{I}-c\hat{\mathbf{A}}-c\mathbf{e}_+\mathbf{e}_s')^{-1}) \\ &= \hat{\mathbf{Q}} + b\hat{\mathbf{Q}}\mathbf{e}_+\mathbf{e}_s'\hat{\mathbf{Q}} \\ &= \hat{\mathbf{Q}} + b\mathbf{u}\hat{\mathbf{Q}}(s,:) \end{aligned} \tag{4.10}$$

Table 4.2: Summary of data sets

dataset	number of nodes	number of edges
AC	421,807	1,066,816
ML	4,563	20,469
CoMMG	54,200	354,186

where vector $\mathbf{u}$ is defined as in step 24 and the scale b satisfies

$$\begin{aligned} b &= \frac{c}{1-c-c\mathbf{e}_s'\hat{\mathbf{Q}}\mathbf{e}_+} \\ &= \frac{c}{1-c-c\mathbf{e}_s'\mathbf{u}} \\ &= \frac{c}{1-c-c\mathbf{u}(s)} \end{aligned} \tag{4.11}$$

Putting equations (6.6), (6.8) and (4.11) together, we have that the correct ranking vector for the source node s on matrix $\tilde{\mathbf{A}}$ must satisfies:

$$\begin{aligned} \tilde{\mathbf{Q}}(:,s) &= \hat{\mathbf{Q}}(:,s) + b\mathbf{u}\hat{\mathbf{Q}}(s,s) \\ &= \hat{\mathbf{r}}_s + \frac{c\hat{\mathbf{r}}_s(s)}{1-c-c\mathbf{u}(s)}\mathbf{u} \\ &= \tilde{\mathbf{r}}_s \end{aligned} \tag{4.12}$$

where $\tilde{\mathbf{r}}_s$ is defined as in step 25, which completes the proof of theorem 1. □

4.5 Experimental Evaluations

In this section we present experimental results. All the experiments are designed to answer the following questions:

- *Effectiveness:* What data mining observations does the proposed iPoG enable?
- *Efficiency:* How does the proposed Fast-iPoG balance between speed and quality?

4.5.1 Experimental Setup

Data Sets. We use three datasets in our experiments, which are summarized in Table 4.2.

The first data set (*AC*) is from DBLP.[4] It is an author-conference bipartite graph, where each row corresponds to an author and each column corresponds to a conference. An edge weight is the number of papers that the corresponding author publishes in the corresponding conference. On the whole, there are 421,807 nodes (418,236 authors and 3,571 conferences) and 1,066,816 edges in the graph.

[4] `http://www.informatik.uni-trier.de/~ley/db/`

The second data set (*ML*) uses author-paper information from two major machine learning conferences ('NIPS', and 'ICML') to construct a co-authorship graph, where each node represents an author and an edge weight is the number of co-authored papers between any two corresponding authors. On the whole, there are 4,563 nodes and 20,469 edges.

The third data set (*CoMMG*) is used in [PYFD04], which contains around 7,000 captioned images, each with about 4 captioned terms. There are in total 160 terms for captioning. In our experiments, 1,740 images are set aside for testing. The graph matrix is constructed exactly as in [PYFD04], which contains 54,200 nodes and 354,186 edges.

Parameter Settings. There are two parameters in the proposed iPoG: c for random walk with restart, and k for the neighborhood size of a given negative node. We set $c = 0.95$ (as suggested in [TFP08]). To determine k, a parametric study has been performed[5] and ProSin shows little sensitivity to k for a large range of settings (from $k = 2$ to $k = 10$). For the experimental results in this paper, k is set to be 5.

Machine Configurations. For the computational cost, we report the wall-clock time. All the experiments ran on the same machine with four 3.0GHz Intel (R) Xeon (R) CPUs and 16GB memory, running Linux (2.6 kernel). For each experiment, we run it 10 times and report the average.

4.5.2 Effectiveness: Case Studies

In both the proposed iPoG and the original random walk with restart, the proximity score is defined as the steady-state probability . Therefore, we expect it to enrich a broad range of applications by replacing the original random walk with restart with our iPoG. In this subsection, we present three applications as case studies: neighborhood search, center-piece subgraphs, and image caption.

Neighborhood Search. By incorporating the users' feedback, we can allow interactive neighborhood search on the graph. Fig. 4.4 gives one such example, where we want to find the top 10 neighbors of 'KDD' conferences (i.e, the 10 most similar conferences as 'KDD') from the *AC* data set. In Fig. 4.4(a), we plot the initial results when there is no side information (i.e, $\mathcal{P} = \phi$ and $\mathcal{N} = \phi$). Subjectively, the result makes sense, which reflects two major sub-communities in 'KDD': the AI/statistic community (e.g., 'ICML', 'NIPS', and 'IJCAI') and the databases community (e.g., 'SIGMOD', 'VLDB', 'ICDE' etc). Then, if the user gives negative feedback on 'ICML' (i.e, $\mathcal{P} = \phi$ and $\mathcal{N} = \{'\mathrm{ICML}'\}$), all the AI/statistic related conferences ('NIPS' and 'IJCAI') disappear (See Fig. 4.4(b)). In Fig. 4.4(c), we present the updated result if the user further gives some positive feedback on 'SIGIR', which is one of the major conferences on information retrieval. Again, the result confirms the effectiveness of ProSIN: positive feedback on 'SIGIR' brings more information retrieval related conferences (e.g, 'TREC', 'CIKM', 'ECIR','CLEF', 'ACL', 'JCDL', etc).

Center-Piece Subgraphs. The concept of connection subgraphs, or center-piece subgraphs, was proposed in [FMT04, TF06]: Given Q query nodes, it creates a subgraph $\mathcal{H}$ that shows the relationships between the query nodes. The resulting subgraph should contain the nodes that have strong connection to all or most of the query nodes. Moreover, since this subgraph $\mathcal{H}$ is used for

[5]We skip the details of the parametric study for brevity.

visually demonstrating node relations, its visual complexity is capped by setting an upper limit, or a *budget* on its size. These so-called connection subgraphs (or center-piece subgraphs) were proved useful in various applications, but currently cannot handle users' interaction (i.e, feedback).

One of the building block in the original center-piece subgraphs [TF06] is to use RWR to measure the proximity from the query nodes to the remaining nodes on the graph. Therefore, by replacing the original random walk with restart by the proposed iPoG, we can naturally deal with the users' interactions (for details of center-piece subgraph, please refer to [TF06]).

Fig. 4.5 plots an example to find the center-piece subgraphs between two researchers ('Andrew Mccallum' and 'Yiming Yang') from *ML* data set. In Fig. 4.5(a), we plot the initial results when there is no side information (i.e, $\mathcal{P} = \phi$ and $\mathcal{N} = \phi$). It can be seen that there are two major connections between 'Andrew Mccallum' and 'Yiming Yang': one connection is on text mining/information retrieval (through 'Rebecca Hutchinson', 'Xuerui Wang', 'Tom M. Mitchell', 'Sean Slattery' and 'Rayid Ghani'), and the other connection in on AI/statistics (throught 'John D. Laffterty', 'Zoubin Ghahramani' and 'Jian Zhang'). Fig. 4.5(b) gives the updated result if the user gives negative feedback on 'Tom M. Mitchell'. It can be seen that the whole connection on text mining/information retrieval disappears, and more connection on AI/statistics (e.g. through 'Andrew Ng' and 'Michael I. Jordan') shows up.

Image Caption. Here, the goal is to assign some keywords for a given image as its text annotation. In [PYFD04], the authors proposed a graph based solution and showed its superiority over the traditional methods in feature space. The key idea of [PYFD04] is to construct an image-keyword-region graph and use RWR to measure the relevance between the test image and the known keywords. Similar to center-piece subgraphs, replacing RWR by iPoGcan easily incorporate side information (if available) in such process.

Fig. 4.6 presents the average precison/recall on *CoMMG* data set. Here, the side-information is simulated as following: for each test image, 5 keywords that are most relevant to the test im-

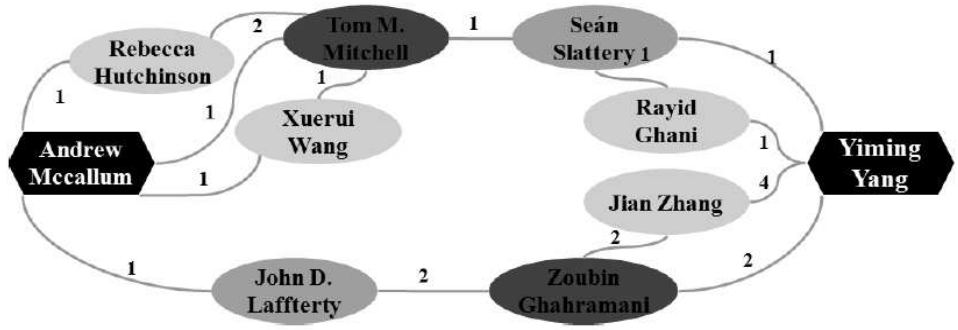
'ICDM' 'ICML' 'SDM' 'VLDB' 'ICDE' 'SIGMOD' 'NIPS' 'PKDD' 'IJCAI' 'PAKDD'	'ICDM' 'SDM' 'PKDD' 'ICDE' 'VLDB' 'SIGMOD' 'PAKDD' 'CIKM' 'SIGIR' 'WWW'	'SIGIR' 'TREC' 'CIKM' 'ECIR' 'CLEF' 'ICDM' 'JCDL' 'VLDB' 'ACL' 'ICDE'
(a) No feedback	(b) $\mathcal{N} = \{$'ICML'$\}$	(c) $\mathcal{N} = \{$'ICML'$\}$ $\mathcal{P} = \{$'SIGIR'$\}$

Figure 4.4: Interactive neighborhood search for 'KDD' conference.

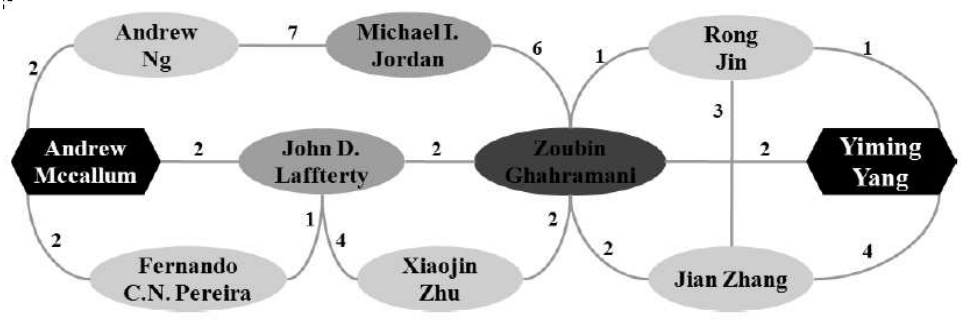
age based on the current proximity measurement are returned for users' yes/no (i.e., correct/wrong caption) confirmation. Here, we also compare two simple strategies: (1) 'RemNeg', where the negative nodes are simply removed from the graph; and (2) 'LinCom' [HLZ+04], where the proximity scores from positive/negative nodes are added/substracted from the score from the test image. From the figure, it can be seen that our iPoG largely improves both precision/recall for image caption task by incorporating such side information. For example, it improves the precision by 13.59% (44.02% vs. 30.43%) and the recall by 17.39% (57.54% vs. 40.15%) when the prediction length is 4. It is interesting to notice that if we simply remove the negative nodes from the graph, it will actually hurts the performance ('RemNeg'). As for 'LinCom', it can be seen that (1) the improvement is limited compared with the proposed iPoG for short prediction length; and (2) it might hurt the performance with the increase of the prediction length.

4.5.3 Efficiency

In this subsection, we study the quality/speed trade-off of the proposed Fast-iPoG. We use the *CoMMG* data set (since it is the only one with ground truth among the three data sets we used in this chapter). Here, we fix the prediction length to be 4 (the results with other prediction length are similar and therefore skipped for brevity), and we compare the precision/recall between Fast-



(a) No feedback

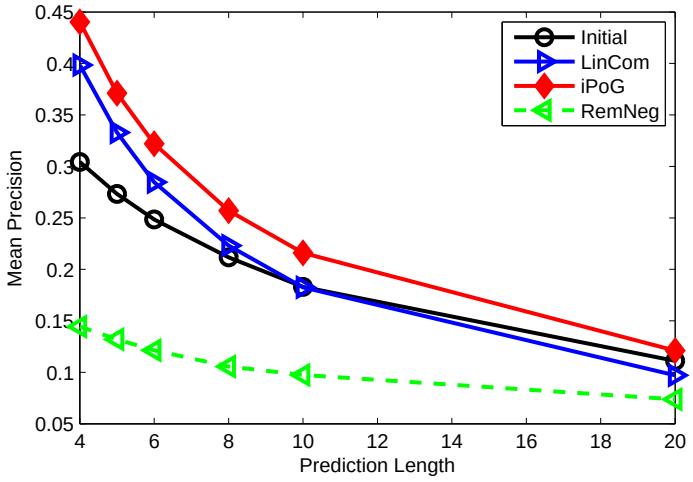


(b) Negative feedback on 'Tom M. Mitchell'

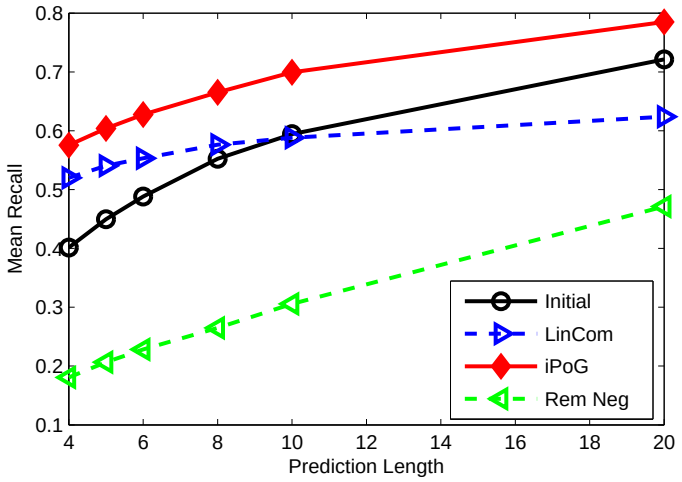
Figure 4.5: Interactive center-piece subgraphs between 'Andrew Mccallum' and 'Yiming Yang'.

iPoG and iPoG where in iPoG random walk with restart is performed by the iterative method.[6] Compared with iPoG, there is one more parameter in Fast-iPoG, the rank of the low-rank approximation for $\mathrm{NB\_LIN\_Pre}()$. We vary this parameter from 100 to 600 (denoted as Fast-iPoG(100), Fast-iPoG(200), etc in Fig. 4.7). In order to put quality/speed in the same figure, we normalized

[6]An alternative choice for iPoG is to run $\mathrm{NB\_LIN\_Pre}()$ on $\mathbf{A}$ and $\tilde{\mathbf{A}}$ respectively. However, we find it needs more wall-clock time but leads to lower quality compared with the iterative method. Therefore, we only compare the proposed Fast-iPoG with that by iterative method.



(a) Mean precision



(b) Mean recall

Figure 4.6: Incorporate side information for image caption.

Figure 4.7: Quality/speed trade-off of Fast-iPoG.

(1) precision/recall by the largest value for iPoG, and (2) time by the longest value for iPoG.

From Fig. 4.7, it can be seen that the proposed Fast-iPoG achieves significant speedup while maintaining high quality. For example, Fast-iPoG(100) is 49x faster than iPoG (the most right one) while it preserves 93.6% precision (41.2% vs. 44.0%) and 94.0% recall (54.1% vs. 57.5%); Fast-iPoG(400) is 16x faster than iPoG while preserving 96.1% precision (42.4% vs. 44.0%) and 96.7% recall (55.6% vs. 57.5%). Overall, Fast-iPoG is 10∼49x faster than iPoG, while preserving more than 93.0% quality (for both precision and recall). Note that in all cases, Fast-iPoG significantly improves the precision/recall when compared with the initial case (the left-most dashed bar). As for the wall-clock time, iPoG need 3.7 hours to annotate all the 1,740 images, while Fast-iPoG(100) only needs 4.5 minutes.

4.6 Related Work

In this section, we review the related work, which can be categorized into two parts (1) node proximity and (2) matrix low rank approximation.

Node Proximity. One of the most popular proximity measurements is random walk with restart [HLZ+04, PYFD04, TFP08], which is the baseline of iPoG. Other representative proximity measurements include the sink-augmented delivered current [FMT04], cycle free effective conductance [KNV06], survivable network [GMS93], and direction-aware proximity [TKF07]. All these methods only consider the graph link structure and ignore the side information. Although we focus on random walk with restart in this chapter, our approach (i.e., to use the side information to refine the graph structure) can be applied to other random walk-based measurements, such as [FMT04, TKF07]. In term of dealing with the side information on ranking, our work is also related to [ACA06a], where the goal is to use partial order information to learn the weights of different types of edges. In term of computation, the fast algorithm (NB_LIN) for random walk with restart in [TFP08] is most related to the proposed Fast-iPoG. Our Fast-iPoG differs from that in [TFP08] in the sense that the graph structure in our setting keeps changing by the side information, whereas it is fixed in [TFP08]. The core idea behind the proposed Fast-iPoG is to leverage the smoothness between graph structure with/without side information. In [TPYF08], the authors has used the similar idea to track the proximity/centrality on a time-evolving skewed bipartite graph. Other remotely related work includes [GKRT04], where the goal is to propagate the trust/distrust to predict the trust between any two persons.

Graph proximity is an important building block in many graph mining settings. Representative work includes connection subgraphs [FMT04, KNV06, TF06], neighborhood search in bipartite graphs [SQCF05], content-based image retrieval [HLZ+04], cross-modal correlation discovery [PYFD04], the BANKS system [ABC+02], link prediction [LNK03], pattern matching [TFGER07], ObjectRank [BHP04], RelationalRank [GMT04] and recommendation system [CTSP07]. Note that for the ranking-related tasks (such as neighborhood search, image retrieval, etc.), we can also use the linear combination strategy suggested in [HLZ+04]; the strategy includes personalized PageRank [Hav03] and graph-based semi-supervised learning [ZBL+03] as a special case when the negative set is absent, to incorporate like/dislike type of side information. Our experimental evaluation on image caption task shows that although it is effective for small prediction lengths, its

performance is not as good as the proposed iPoG and sometimes it actually hurts the performance. What is more important, it is not clear how to use such strategy (linear combination) for more complicated applications (such as center-piece subgraphs, pattern match etc). This is exactly one major advantage of the proposed iPoG: it can be easily plugged into such applications by simply replacing the original proximity measurement by our iPoG.

Low Rank Approximation. Low rank approximation [GVL89, DKM05b, AM07] plays a very important role in graph mining. Please refer to Chapter 2 for details. Notice that our Fast-iPoG is orthogonal to the specific method of low rank approximation.

4.7 Conclusion and Discussion

Summary of This Chapter. In this chapter, we study how to incorporate like/dislike type of side information in measuring node proximity on large graphs. Our main contributions are in two folds. First, we proposed a novel method (iPoG) to incorporate side information in measuring node proximity on large graphs and showed its broad applicability through various case studies. Second, to enhance the efficiency of iPoG, we also took advantage of the smoothness of the graph structures with/without side information and proposed a fast algorithm (Fast-iPoG). We demonstrated that Fast-iPoG achieves significant speedup (up to 49x) in our evaluation on real datasets. Overall, we expect the proposed algorithms to enrich a broad range of applications that receive online feedback/side information.

Discussions. In this chapter, we have focused on the uni-partite graphs and we have *empirically* show the superiority of the proposed iPoG. In [TQJF09], we have generalized this work in two dimensions: (1) we show that the proposed iPoG actually does adaptive linear combination, which explains why we would expect it performs better than alternative choices; (2) we proposed a fast algorithm for bi-partite graphs, which achieves orders of magnitude speedup with *no quality loss*.

Chapter 5

Case Study #3: Gateway

Summary of This Chapter

- **Questions we want to answer:**

  Q: What is the best gateway between a source node (or source group) and a target node (or target group), in a network?

- **Our answers and contributions**

  A1: We proposed a novel gateway-ness score for a given source and target, that agrees with human intuition. We generalize it to the case where we have a group of nodes as the source and the target.

  A2: We proposed two algorithms to find a set of nodes with the highest gateway-ness score, which (1) are fast and scalable; and (2) lead to near-optimal results.

5.1 Introduction

What is the best gateway between a source node and a target node, in a network? This is a core problem that appears under several guises, with numerous generalizations. Motivating applications include the following:

1. In a corporate social network, which are the key people that bring or hold different groups together? Or, if seeking to establish a cross-division project, who are the best people to lead such an effort?
2. In an immunization setting, given a set of nodes that are infected, and a set of nodes we want to defend, which are the best few 'gateways' we should immunize?
3. Similarly, in a network setting, which are the gateway nodes we should best defend against an attack, to maximize connectivity from source to target.
4. Given a graph of co-workers and their skills (keywords), whom should you contact to learn

more about, say, Linux? You want someone reasonably close to you and fairly well-versed in Linux, but not your secretary or Linus Torvalds himself.

The problem has several, natural generalizations: (a) we may be interested in the top k best gateways (in case our first few choices are unavailable); (b) we may have more than one source nodes, and more than one target nodes, as in the immunization setting above; (c) we may have a bi-partite graph with relationships (edges) between different node types, as in the last example above. Our main contributions in this chapter are:

- A novel 'gateway-ness' score for a given source and target, that agrees with human intuition. Its generalization to the case where we have a group of nodes as the source and the target;
- Two algorithms to find a set of nodes with the highest 'gateway-ness' score, which (1) are fast and scalable; and (2) lead to *near-optimal* results;
- Extensive experimental results on real data sets, showing the effectiveness and efficiency of the proposed methods.

The rest of the chapter is organized as follows: We give the problem definitions in Section 2; present 'gateway-ness' scores in Section 3; and deal with the computational issues in Section 4. We evaluate the proposed methods in Section 5. Finally, we review the related work in Section 6 and conclude in Section 7.

5.2 Problem Definitions

Table 5.1 lists the main symbols we use throughout this chapter. Here, we focus on directed weighted graphs. We represent the graph by its normalized adjacency matrix ($\mathbf{A}$). Following standard notation, we use capital bold letters for matrices (e.g., $\mathbf{A}$), lower-case bold letters for vectors (e.g., $\mathbf{a}$), and calligraphic fonts for sets (e.g., $\mathcal{S}$). We denote the transpose with a prime (i.e., $\mathbf{A}'$ is the transpose of $\mathbf{A}$). We use arrowed lower-case letters for paths on the graph (e.g., $\vec{p}$), which are ordered sequences. We use parenthesized superscripts to represent source/target information for the corresponding variables. For example $\vec{p}^{(s,t)} = \{s = u_0, u_1, ..., u_l = t\}$ is a path from the source node s to the target node t. If the source/target information is clear from the context, we omit the superscript for brevity. A sink node i on the graph is a node without out-links (i.e., $\mathbf{A}(:, i) = 0$). We use subscripts to denote the corresponding variable after setting the nodes indexed by the subscripts as sinks. For example, $\vec{p}_{\mathcal{I}}^{(s,t)}$ is the path from the source node s to the target node t, which does not go through any nodes indexed by the set $\mathcal{I}$ (i.e., $u_i \notin \mathcal{I}, i = 0, ..., l$). With the above notations, our problems can be formally defined as follows:

Problem 3. (Pair-Gateway)

Given: *a weighted directed graph* $\mathbf{A}$*, a source node* s*, a target node* t*, and a budget (integer)* k*;*

Find: *a set of at most* k *nodes which have the highest 'gate-way-ness' score wrt the source node* s *and the target node* t*.*

Problem 4. (Group-Gateway)

Given: *a weighted directed graph* $\mathbf{A}$*, a group of source nodes* $\mathcal{S}$*, a group of target nodes* $\mathcal{T}$*, and a budget (integer)* k*;*

Table 5.1: Symbols

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}, \ldots$	matrices (bold upper case)
$\mathbf{A}(i, j)$	the element at the i^{th} row and j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}(i, :)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	the j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{a}, \mathbf{b}, \ldots$	column vectors
$\vec{p}, \vec{q}, \ldots$	ordered sequences
$\mathcal{S}, \mathcal{T}, \ldots$	sets (calligraphic)
n	number of nodes in the graph
m	number of edges in the graph
$\mathrm{g}(s, t, \mathcal{I})$	the 'Gateway-ness' score for the subset of nodes $\mathcal{I}$ wrt the source s and the target t
$\mathrm{g}(\mathcal{S}, \mathcal{T}, \mathcal{I})$	the 'Gateway-ness' score for the subset of nodes $\mathcal{I}$ wrt the source group $\mathcal{S}$ and the target group $\mathcal{T}$
$\mathbf{r}(s, t)$	the proximity score from s to t
$\mathbf{r}_{\mathcal{I}}(s, t)$	the proximity score from s to t by setting the subset of nodes indexed by $\mathcal{I}$ as sinks

Find: *a set of at most k nodes which have the highest 'gate-way-ness' score wrt the source group $\mathcal{S}$ and the target group $\mathcal{T}$.*

In both Problem 3 (Pair-Gateway) and Problem 4 (Group-Gateway), there are two sub-problems: (1) how to define the 'gateway-ness' score of a given subset of nodes $\mathcal{I}$; (2) how to find the subset of nodes with the highest 'gateway-ness' score. In the next two sections, we present the solutions for each, respectively.

5.3 Proposed 'Gateway-ness' Scores

In this section, we present our definitions for 'Gateway-ness'. We first focus on the case of a single source s and a single target t (Pair-Gateway). We then generalize to the case where both the source and the target are a group of nodes (Group-Gateway)

5.3.1 Node 'Gateway-ness' Score

Given a single source s and a single target t, we want to measure the 'Gateway-ness' score for a given set of nodes $\mathcal{I}$. We first give the formal definitions in such a setting and then provide some intuitions for our definitions.

Formal Definitions. For a graph $\mathbf{A}$, we can use random walk with restart to measure the proximity (i.e., relevance/closeness) from the source node s to the target node t, which is defined as follows: Consider a random particle that starts from node s. The particle iteratively transits to its neighbors with probability proportional to the corresponding edge weights. Also at each step, the particle returns to node s with some restart probability $(1-c)$. The proximity score from node s to node t is defined as the steady-state probability $\mathbf{r}(s,t)$ that the particle will be on node t [TFP08]. Intuitively, $\mathbf{r}(s,t)$ is the fraction of time that the particle starting from node s will spend on node t of the graph, after an infinite number of steps.

Intuitively, a set of nodes $\mathcal{I}$ are good gateways wrt s and t if they play an important role in the proximity measure from the source to the target. Therefore, our 'Gateway-ness' score can be defined as follows:

$$\mathrm{g}(s,t,\mathcal{I}) \triangleq \Delta\mathbf{r}(s,t) \triangleq \mathbf{r}(s,t) - \mathbf{r}_{\mathcal{I}}(s,t) \tag{5.1}$$

where $r_{\mathcal{I}}(s,t)$ is the proximity score from source s to t after setting the subset of nodes indexed by $\mathcal{I}$ as sinks.

Intuitions. Here, we provide some intuition of the 'Gateway-ness' score defined by eq.(5.1), using the running example in figure 5.1.

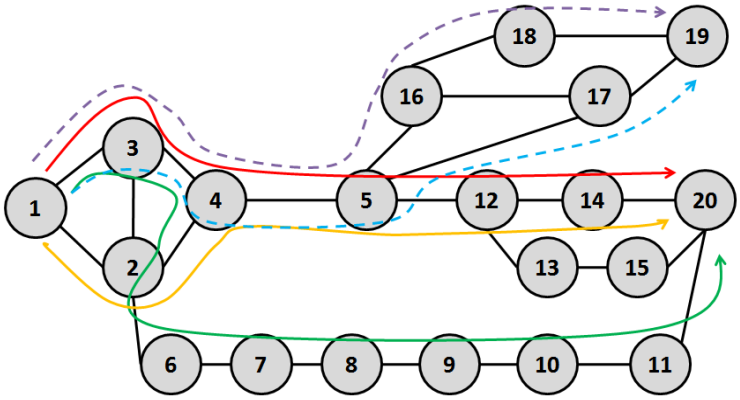


Figure 5.1: Running example (best viewed in color)

In figure 5.1, each solid arrowed line is a path from node 1 to node 20, which can be denoted by an ordered sequence. For example, the path marked by the red line can be denoted by $\vec{p}^{(1,20)} = \{1,3,4,5,12,14,20\}$. For each path $\vec{p}^{(s,t)} = \{s = u_0, u_1, ..., u_l = t\}$, we can define its score by eq (5.2), where $\prod_{i=0}^{l} \mathbf{A}(u_{i-1}, u_i)$ is the probability that the random particle will traverse this path, and $(1-c)c^l$ penalizes the length of the path. For example, the red path ($\vec{p}^{(1,20)} = \{1,3,4,5,12,14,20\}$), has score $(1-c)c^6\mathbf{A}(3,1)\mathbf{A}(4,3)\mathbf{A}(5,4)\mathbf{A}(12,5)\ \mathbf{A}(14,12)\mathbf{A}(20,14)$.

$$\text{score}(\vec{p}^{(s,t)}) \triangleq (1-c)c^l \prod_{i=0}^{l} \mathbf{A}(u_{i-1}, u_i) \tag{5.2}$$

where $\mathbf{A}$ is the normalized adjacency matrix of the graph.

With the above definitions for the path score, we have the following lemma:

Lemma 5. Sum of Weighted Path Scores. *Let $\vec{P}$ be the set of all the paths from the source node s to the target node t, and $\vec{Q}$ be the set of all the paths from the source node s to the target node t which go through at least one node indexed by the subset $\mathcal{I}$. Let $\mathbf{r}(s,t)$ be the proximity score defined by random walk with restart and $g(s,t,\mathcal{I})$ be the 'Gateway-ness' score defined by eq. (5.1). Then we have*

$$\begin{aligned} \mathbf{r}(s,t) &= \sum_{\vec{p}^{(s,t)} \in \vec{P}} \mathit{score}(\vec{p}^{(s,t)}) \\ g(s,t,\mathcal{I}) &= \sum_{\vec{p}^{(s,t)} \in \vec{Q}} \mathit{score}(\vec{p}^{(s,t)}) \end{aligned} \tag{5.3}$$

Proof: By induction, we can verify that

$$(1-c)(c\mathbf{A})^k(t,s) = \sum_{\vec{p}^{(s,t)} \in \vec{P};\ \text{length of } \vec{p}^{(s,t)} = k} \text{score}(\vec{p}^{(s,t)}) \ (k = 1, 2, 3, ...) \tag{5.4}$$

In other words, $(1-c)(c\mathbf{A})^k(t,s)$ accounts for the sum of scores of all the paths from s to t with length k.

On the other hand, by Taylor expansion, we have

$$\begin{aligned} \mathbf{Q} &= (1-c)(\mathbf{I} - c\mathbf{A})^{-1} \\ &= (1-c)\sum_{k=0}^{\infty}(c\mathbf{A})^k \end{aligned} \tag{5.5}$$

Since $s \neq t$, we have

$$\begin{aligned} \mathbf{r}(s,t) &= \mathbf{Q}(t,s) \\ &= (1-c)\sum_{k=0}^{\infty}(c\mathbf{A})^k(t,s) \\ &= (1-c)\sum_{k=1}^{\infty}(c\mathbf{A})^k(t,s) \\ &= \sum_{\vec{p}^{(s,t)} \in \vec{P}} \text{score}(\vec{p}^{(s,t)}) \end{aligned} \tag{5.6}$$

Similarly, we can show that

$$\mathbf{r}_{\mathcal{I}}(s,t) = \sum_{\vec{p}^{(s,t)} \in \vec{P}/\vec{Q}} \text{score}(\vec{p}^{(s,t)}) \tag{5.7}$$

Therefore,

$$\begin{aligned} \mathrm{g}(s,t,\mathcal{I}) &\triangleq \mathbf{r}(s,t) - \mathbf{r}_{\mathcal{I}}(s,t) \\ &= \sum_{\vec{p}^{(s,t)} \in \vec{Q}} \text{score}(\vec{p}^{(s,t)}) \end{aligned} \tag{5.8}$$

which completes the proof. □

By eq. (5.3), the 'Gateway-ness' score for a given set of nodes $\mathcal{I}$ accounts for all the paths from the source node s to the target node t which pass through one or more nodes in $\mathcal{I}$. For example, given the source node 1 and the target node 20 in figure 5.1, the 'Gateway-ness' score for $\mathcal{I} = \{2\}$ is the sum of the scores of all the paths from node 1 to node 20 that go through node 2 (e.g., the green path, the yellow path, and so on).

5.3.2 Group 'Gateway-ness' Score

Here we consider the case where the source and/or target consist of more than one nodes. Suppose we have a group of source nodes $\mathcal{S}$ and a group of target nodes $\mathcal{T}$. Then, the 'Gateway-ness' score for a given set of nodes $\mathcal{I}$ can be defined in a similar way:

$$\mathrm{g}(\mathcal{S}, \mathcal{T}, \mathcal{I}) \triangleq \sum_{s \in \mathcal{S}, t \in \mathcal{T}} \Delta \mathbf{r}(s,t) \triangleq \sum_{s \in \mathcal{S}, t \in \mathcal{T}} (\mathbf{r}(s,t) - \mathbf{r}_{\mathcal{I}}(s,t)) \tag{5.9}$$

where $r_{\mathcal{I}}(s,t)$ is the proximity score from s to t by setting the subset of nodes indexed by $\mathcal{I}$ as sinks (i.e., delete all out-edges, by setting $\mathbf{A}(:,i) = 0$ for all $i \in \mathcal{I}$).

Intuitively, the score defined by eq. (5.9) accounts for all the paths from the source group to the target group[1] which go through at least one node in $\mathcal{I}$. For example, given $\mathcal{S} = \{1\}$ and $\mathcal{T} = \{19, 20\}$ in figure 5.1, the group 'Gateway-ness' score for $\mathcal{I} = \{5, 8\}$ corresponds to all the paths from node 1 to 19 or 20 (e.g., red, yellow and green solid lines, purple and blue dashed lines and so on).

5.4 BASSET: Proposed Fast Solutions

In this section, we address how to quickly find a subset of nodes of the highest 'Gateway-ness' score. We start by showing that the straight-forward methods (referred to as 'Com-RWR') are computationally intractable. Then, we present the proposed BASSET (BASSET-N for Pair-Gateway and BASSET-G for Group-Gateway). For each case, we first present the algorithm and then analyze its effectiveness as well as its computational complexity.

5.4.1 Computational Challenges

Here, we present the computational challenges and the way we tackle them. For the sake of succinctness, we mainly focus on BASSET-N.

There are two main computational challenges in order to find a subset of nodes with the highest 'Gateway-ness' score. First of all, we need to compute the proximity from the source to the target on different graphs, each of which is a perturbed version of the original graph. This essentially means that we cannot directly apply some powerful pre-computational method to evaluate the

[1] A path from the source group to the target group is a path which starts from a node of the source group and ends at a node of the target group.

proximity from the source to the target (after setting the subset of nodes indexed by $\mathcal{I}$ as sinks). Instead, we have to rely on on-line iterative methods, whose computational complexity is $O(m)$. The challenges are compounded by the need to evaluate $\mathrm{g}(s,t,\mathcal{I})$ (eq. (5.1)) or $\mathrm{g}(\mathcal{S},\mathcal{T},\mathcal{I})$(eq. (5.9)) an exponential number of times ($\binom{n}{k}$). Putting these together, the straightforward way to find k nodes with the highest 'Gateway-ness' score is $O(\binom{n}{k}m)$. This is computationally intractable. Suppose on a graph with $1,000,000$ nodes, we want to find the best $k=5$ gateway nodes. If computing each proximity score takes 0.001 seconds, then 2.64×10^{17} years are needed to find the gateways. This is much longer than the age of the universe.[2]

To tackle such challenges, we resort to two main ideas, which are summarized in Theorem 2. According to Theorem 2, in order to evaluate the 'Gateway-ness' score of a given set of nodes, we do not need to actually set these nodes as sinks and compute the proximity score on the new graph. Instead, we can compute it from the original graph. In this way, we can utilize methods based on pre-computation to accelerate the process. Furthermore, since $\mathrm{g}(s,t,\mathcal{I})$ and $\mathrm{g}(\mathcal{S},\mathcal{T},\mathcal{I})$ are sub-modular wrt $\mathcal{I}$, we can develop some greedy algorithm to avoid exponential enumeration, and still get some *near-optimal* solution. In Theorem 2, $\mathbf{A}$ is the normalized adjacency matrix of the graph. It is worth pointing out that The proposed methods (BASSET-N and BASSET-G) we will introduce are orthogonal to the specific way of normalization. For simplicity, we use column-normalization throughout this chapter. Also, $\mathbf{Q}(\mathcal{I},\mathcal{I})$ is a $|\mathcal{I}|\times|\mathcal{I}|$ matrix, containing the elements in the matrix $\mathbf{Q}$ which are at the rows/columns indexed by $\mathcal{I}$. Similarly, $\mathbf{Q}(t,\mathcal{I})$ is a row vector with length $|\mathcal{I}|$, containing the elements in the matrix $\mathbf{Q}$ which are at the t^{th} row and the columns indexed by $\mathcal{I}$. $\mathbf{Q}(\mathcal{I},s)$ is a column vector with length $|\mathcal{I}|$, containing the elements in the matrix $\mathbf{Q}$ which are at the s^{th} column and the rows indexed by $\mathcal{I}$.

Theorem 2. Core Theorem. *Let $\mathbf{A}$ be the normalized adjacency matrix of the graph, and $\mathbf{Q}=(1-c)(\mathbf{I}-c\mathbf{A})^{-1}$. For a given source s and target t, the 'Gateway-ness' score of a subset of nodes $\mathcal{I}$ defined in eq. (5.1) satisfies the properties P1 and P2. For a given source group $\mathcal{S}$ and target group $\mathcal{T}$, the 'Gateway-ness' score of a subset of nodes $\mathcal{I}$ defined in eq. (5.9) satisfies the properties P3 and P4, where $s\neq t$, $s,t\notin\mathcal{I}$, $\mathcal{S}\bigcap\mathcal{T}=\emptyset$, $\mathcal{S}\bigcap\mathcal{I}=\emptyset$, and $\mathcal{T}\bigcap\mathcal{I}=\emptyset$.*

P1. $g(s,t,\mathcal{I})=\mathbf{Q}(t,\mathcal{I})\mathbf{Q}(\mathcal{I},\mathcal{I})^{-1}\mathbf{Q}(\mathcal{I},s)$;

P2. $g(s,t,\mathcal{I})$ is sub-modular wrt the set $\mathcal{I}$, that is, $g(s,t,\mathcal{I}\bigcup\mathcal{J})+g(s,t,\mathcal{I}\bigcap\mathcal{J})\leq g(s,t,\mathcal{I})+g(s,t,\mathcal{J})$, for any subsets $\mathcal{I}$ and $\mathcal{J}$;

P3. $g(\mathcal{S},\mathcal{T},\mathcal{I})=\sum_{s\in\mathcal{S},t\in\mathcal{T}}\mathbf{Q}(t,\mathcal{I})\mathbf{Q}(\mathcal{I},\mathcal{I})^{-1}\mathbf{Q}(\mathcal{I},s)$;

P4. $g(\mathcal{S},\mathcal{T},\mathcal{I})$ is sub-modular wrt the set $\mathcal{I}$, that is, $g(\mathcal{S},\mathcal{T},\mathcal{I}\bigcup\mathcal{J})+g(\mathcal{S},\mathcal{T},\mathcal{I}\bigcap\mathcal{J})\leq g(\mathcal{S},\mathcal{T},\mathcal{I})+g(\mathcal{S},\mathcal{T},\mathcal{J})$, for any subsets $\mathcal{I}$ and $\mathcal{J}$.

Proof of P1: WLOG, we assume that $\mathcal{I}=\{n-k+1,...n\}$. Let $\mathbf{A}$ and $\tilde{\mathbf{A}}$ be the normalized adjacency matrices of the graph before/after we set the subset of nodes in $\mathcal{I}$ as sinks. Write $\mathbf{A}$ and $\tilde{\mathbf{A}}$ in block form:

$$\mathbf{A}=\begin{pmatrix}\mathbf{A}_{1,1} & \mathbf{A}_{1,2}\\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2}\end{pmatrix},\ \tilde{\mathbf{A}}=\begin{pmatrix}\tilde{\mathbf{A}}_{1,1} & \tilde{\mathbf{A}}_{1,2}\\ \tilde{\mathbf{A}}_{2,1} & \tilde{\mathbf{A}}_{2,2}\end{pmatrix}=\begin{pmatrix}\mathbf{A}_{1,1} & \mathbf{0}\\ \mathbf{A}_{2,1} & \mathbf{0}\end{pmatrix} \tag{5.10}$$

where $\mathbf{0}$ is a matrix with all zero elements.

[2]According to Wikipedia, (`http://en.wikipedia.org/wiki/`Age_of_the_universe), the age of the universe is about 1.4×10^{10} years.

Let $\tilde{\mathbf{Q}} = (1-c)(\mathbf{I} - c\tilde{\mathbf{A}})^{-1}$. We can also write $\tilde{\mathbf{Q}}$ and $\mathbf{Q}$ in block form:

$$\begin{aligned}
\mathbf{Q} &= (1-c)(\mathbf{I} - c\mathbf{A})^{-1} = \begin{pmatrix} \mathbf{Q}_{1,1} & \mathbf{Q}_{1,2} \\ \mathbf{Q}_{2,1} & \mathbf{Q}_{2,2} \end{pmatrix} \\
&= (1-c)\begin{pmatrix} \mathbf{I} - c\mathbf{A}_{1,1} & -c\mathbf{A}_{1,2} \\ -c\mathbf{A}_{2,1} & \mathbf{I} - c\mathbf{A}_{2,2} \end{pmatrix}^{-1} \\
\tilde{\mathbf{Q}} &= \begin{pmatrix} \tilde{\mathbf{Q}}_{1,1} & \tilde{\mathbf{Q}}_{1,2} \\ \tilde{\mathbf{Q}}_{2,1} & \tilde{\mathbf{Q}}_{2,2} \end{pmatrix} = (1-c)\begin{pmatrix} \mathbf{I} - c\mathbf{A}_{1,1} & \mathbf{0} \\ -c\mathbf{A}_{2,1} & \mathbf{I} \end{pmatrix}^{-1}
\end{aligned}$$

Applying the block matrix inverse lemma [PC90] to $\tilde{\mathbf{Q}}$ and $\mathbf{Q}$, we get the following equations:

$$\begin{aligned}
\tilde{\mathbf{Q}}_{1,1} &= (1-c)(\mathbf{I} - c\mathbf{A}_{1,1})^{-1}, \;\; \tilde{\mathbf{Q}}_{1,2} = \mathbf{0} \\
\tilde{\mathbf{Q}}_{2,1} &= c(1-c)\mathbf{A}_{2,1}(\mathbf{I} - c\mathbf{A}_{1,1})^{-1}, \;\; \tilde{\mathbf{Q}}_{2,2} = (1-c)\mathbf{I} \\
\mathbf{Q}_{1,1} &= (1-c)(\mathbf{I} - c\mathbf{A}_{1,1})^{-1} + \\
&\quad c^2(\mathbf{I} - c\mathbf{A}_{1,1})^{-1}\mathbf{A}_{1,2}\mathbf{Q}_{2,2}\mathbf{A}_{2,1}(\mathbf{I} - c\mathbf{A}_{1,1})^{-1} \\
\mathbf{Q}_{1,2} &= c(\mathbf{I} - c\mathbf{A}_{1,1})^{-1}\mathbf{A}_{1,2}\mathbf{Q}_{2,2} \\
\mathbf{Q}_{2,1} &= c\mathbf{Q}_{2,2}\mathbf{A}_{2,1}(\mathbf{I} - c\mathbf{A}_{1,1})^{-1}
\end{aligned} \tag{5.11}$$

Therefore, we have

$$\tilde{\mathbf{Q}}_{1,1} = \mathbf{Q}_{1,1} - \mathbf{Q}_{1,2}\mathbf{Q}_{2,2}^{-1}\mathbf{Q}_{2,1} \tag{5.12}$$

On the other hand, based on the properties of random walk with restart [TFP08], we have $\mathbf{r}(i,j) = \mathbf{Q}(j,i)$, and $\mathbf{r}_{\mathcal{I}}(i,j) = \tilde{\mathbf{Q}}(j,i)$,$(i,j = 1,...,n)$. Together with eq. (6.5), we have

$$\begin{aligned}
\mathrm{g}(s,t,\mathcal{I}) &= \mathbf{r}(s,t) - \mathbf{r}_{\mathcal{I}}(s,t) \\
&= \mathbf{Q}_{1,1}(t,s) - \tilde{\mathbf{Q}}_{1,1}(t,s) \\
&= \mathbf{Q}_{1,2}(t,:)\mathbf{Q}_{2,2}^{-1}\mathbf{Q}_{1,2}(:,s)
\end{aligned} \tag{5.13}$$

which completes the proofs of P1. □

Proof of P3: Since P1 holds, we have

$$\begin{aligned}
\mathrm{g}(\mathcal{S},\mathcal{T},\mathcal{I}) &= \sum_{s\in\mathcal{S},t\in\mathcal{T}} \Delta r(s,t) = \sum_{s\in\mathcal{S},t\in\mathcal{T}} \mathrm{g}(s,t,\mathcal{I}) \\
&= \sum_{s\in\mathcal{S},t\in\mathcal{T}} \mathbf{Q}(t,\mathcal{I})\mathbf{Q}(\mathcal{I},\mathcal{I})^{-1}\mathbf{Q}(\mathcal{I},s)
\end{aligned} \tag{5.14}$$

which completes the proofs of P3. □

Proof of P2: Let $\mathcal{I},\mathcal{J},\mathcal{K}$ be three subsets and $\mathcal{I} \subseteq \mathcal{J}$. We will first prove by induction that, for any integer power j, the following inequality holds element-wise.

$$\mathbf{A}_{\mathcal{I}}^j - \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^j \geq \mathbf{A}_{\mathcal{J}}^j - \mathbf{A}_{\mathcal{J}\cup\mathcal{K}}^j \tag{5.15}$$

It is easy to verify the base case (i.e.,$j = 1$) for eq. (5.15) holds. Next, assume that eq. (5.15) holds for $j = 1, ..., j_0$, and we want to prove that it also holds for the case $j = j_0 + 1$:

$$\begin{aligned}
& \mathbf{A}_{\mathcal{I}}^{j_0+1} - \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^{j_0+1} \\
= \; & \mathbf{A}_{\mathcal{I}}^{j_0+1} - \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^{j_0}\mathbf{A}_{\mathcal{I}} + \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^{j_0}\mathbf{A}_{\mathcal{I}} - \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^{j_0+1} \\
= \; & (\mathbf{A}_{\mathcal{I}}^{j_0} - \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^{j_0})\mathbf{A}_{\mathcal{I}} + \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^{j_0}(\mathbf{A}_{\mathcal{I}} - \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}) \\
\geq \; & (\mathbf{A}_{\mathcal{J}}^{j_0} - \mathbf{A}_{\mathcal{J}\cup\mathcal{K}}^{j_0})\mathbf{A}_{\mathcal{I}} + \mathbf{A}_{\mathcal{I}\cup\mathcal{K}}^{j_0}(\mathbf{A}_{\mathcal{J}} - \mathbf{A}_{\mathcal{J}\cup\mathcal{K}}) \\
\geq \; & (\mathbf{A}_{\mathcal{J}}^{j_0} - \mathbf{A}_{\mathcal{J}\cup\mathcal{K}}^{j_0})\mathbf{A}_{\mathcal{J}} + \mathbf{A}_{\mathcal{J}\cup\mathcal{K}}^{j_0}(\mathbf{A}_{\mathcal{J}} - \mathbf{A}_{\mathcal{J}\cup\mathcal{K}}) \\
= \; & \mathbf{A}_{\mathcal{J}}^{j_0+1} - \mathbf{A}_{\mathcal{J}\cup\mathcal{K}}^{j_0+1}
\end{aligned} \tag{5.16}$$

In eq. (5.16), the first inequality holds because of the induction assumption. The second inequality holds because $\mathbf{A}_{\mathcal{I}} \geq \mathbf{A}_{\mathcal{J}} \geq 0$ holds element-wise, and $\mathbf{A}_{\mathcal{I}\cup\mathcal{K}} \geq \mathbf{A}_{\mathcal{J}\cup\mathcal{K}} \geq 0$ holds element-wise.

Since $\tilde{\mathbf{Q}} = (1-c)(\mathbf{I} - c\tilde{\mathbf{A}})^{-1} = (1-c)\sum_{j=0}^{\infty}(c\tilde{\mathbf{A}})^j$, we have

$$\begin{aligned}
& \mathrm{g}(s, t, \mathcal{I}\cup\mathcal{K}) - \mathrm{g}(s, t, \mathcal{I}) \\
= \; & (1-c)\sum_{j=0}^{\infty}((c\mathbf{A}_{\mathcal{I}})^j - (c\mathbf{A}_{\mathcal{I}\cup\mathcal{K}})^j \\
\geq \; & (1-c)\sum_{j=0}^{\infty}((c\mathbf{A}_{\mathcal{J}})^j - (c\mathbf{A}_{\mathcal{J}\cup\mathcal{K}})^j \\
= \; & \mathrm{g}(s, t, \mathcal{J}\cup\mathcal{K}) - \mathrm{g}(s, t, \mathcal{J})
\end{aligned} \tag{5.17}$$

Therefore, $\mathrm{g}(s, t, \mathcal{I})$ is sub-modular, which completes the proof of P2. □

Proof of P4: Since $\mathrm{g}(\mathcal{S}, \mathcal{T}, \mathcal{I}) = \sum_{s\in\mathcal{S}, t\in\mathcal{T}} \mathrm{g}(s, t, \mathcal{I})$ (In other words, $\mathrm{g}(\mathcal{S}, \mathcal{T}, \mathcal{I})$ is a non-negative linear combination of sub-modular functions) , according to the linearity of sub-modular functions [KG05], we have that $\mathrm{g}(\mathcal{S}, \mathcal{T}, \mathcal{I})$ is also sub-modular, which completes the proof of P4. □

Intuition. Here, we provide some intuition why $\mathrm{g}(s, t, \mathcal{I})$ and $\mathrm{g}(\mathcal{S}, \mathcal{T}, \mathcal{I})$ are sub-modular. According to Lemma 5, for a given source s and a given target t, $\mathrm{g}(s, t, \mathcal{I}\cup\mathcal{K}) - \mathrm{g}(s, t, \mathcal{I})$ accounts for the scores of all the paths from s to t, which go through some nodes in $\mathcal{K}$ but none of the nodes in $\mathcal{I}$. Therefore, for a given set $\mathcal{K}$, if we already have a bigger subset $\mathcal{J}$, the additional benefit $(\mathrm{g}(s, t, \mathcal{J}\cup\mathcal{K}) - \mathrm{g}(s, t, \mathcal{J}))$ will be relatively small, compared to the case where we have a smaller subset $\mathcal{I}$ $(\mathrm{g}(s, t, \mathcal{I}\cup\mathcal{K}) - \mathrm{g}(s, t, \mathcal{I}))$. For example, in figure 5.1, let $s = 1$, $t = 20$, and $\mathcal{I} = \{5\}$, $\mathcal{J} = \{2, 5\}$. Then, if we have a new subset $\mathcal{K} = \{8\}$, the additional benefit for subset $\mathcal{I}$ accounts for all the paths from $s = 1$ to $s = 20$ which go through node 8, but not node 5 (e.g., the green path, etc). While the additional benefit for subset $\mathcal{J}$ is 0, since all the paths from $s = 1$ to $t = 20$ which go through node 8 must also go through some node in $\mathcal{J}$ (node 2).

5.4.2 BASSET-N for Problem 3

BASSET-N Algorithm

Our fast solution for Problem 3 is summarized in Alg. 5. In Alg. 5, after initialization (step 1), we first pick a node i_0 with the highest $\frac{\mathbf{r}(s,i)\mathbf{r}(i,t)}{\mathbf{r}(i,i)}$ (step 3). Then, in steps 4-14, we find the rest of the

nodes in a greedy way. That is, in each outer loop, we try to find one more node while keeping the current $\mathcal{I}$ unchanged. According to P1 of theorem 2, $\mathbf{v}(i)$ computed in step 7 is the gateway score for the subset $\mathcal{J}$.[3] If the current subset of nodes $\mathcal{I}$ can completely disconnect the source and the target (by setting them as sinks), we will stop the algorithm (step 12). Therefore, Alg. 5 always returns no more than k nodes. It is worth pointing out that in Alg. 5, all the proximity scores are computed from the original graph $\mathbf{A}$. Therefore, we can utilize some powerful methods based on pre-computation to accelerate the whole process. To name a few, for a medium size graph $\mathbf{A}$ (e.g., a few thousands of nodes), we can pre-compute and store the matrix $\mathbf{Q} = (1-c)(\mathbf{I} - c\mathbf{A})^{-1}$; for large unipartite graphs and bipartite graphs, we can use the NB_LIN and BB_LIN algorithms, respectively [TFP08].

Algorithm 5 BASSET-N

Require: the normalized adjacency matrix $\mathbf{A}$, the source node s, the target node t, the budget k and the parameter c

Ensure: a set of nodes $\mathcal{I}$, where $|\mathcal{I}| \leq k$.

1: initialize $\mathcal{I}$ to be empty.
2: compute the proximity score $\mathbf{r}(s,t)$ from the source node s to the target node t.
3: find $i_0 = \text{argmax}_i \frac{\mathbf{r}(s,i)\mathbf{r}(i,t)}{\mathbf{r}(i,i)}$, where $i = 1, ..., n$ and $i \neq s, i \neq t$. add i_0 to $\mathcal{I}$.
4: **for** $j = 2$ to k **do**
5: **for** $i = 1$ to n, and $i \neq s, i \neq t$ and $i \notin \mathcal{I}$ **do**
6: let $\mathcal{J} = \mathcal{I} \cup i$.
7: compute $\mathbf{v}(i) = \mathbf{r}(\mathcal{J}, t)'\mathbf{r}(\mathcal{J}, \mathcal{J})^{-1}\mathbf{r}(s, \mathcal{J})'$
8: **end for**
9: **if** $\max_i \mathbf{v}(i) \leq \mathbf{r}(s,t)$ **then**
10: find $i_0 = \text{argmax}_i \mathbf{v}(i)$; add i_0 to $\mathcal{I}$.
11: **else**
12: break;
13: **end if**
14: **end for**
15: return $\mathcal{I}$

Analysis of BASSET-N.

In this subsection, we analyze the effectiveness and the efficiency of Alg. 5. First, the effectiveness of the proposed BASSET-N is guaranteed by the following lemma. According to Lemma 6, although BASSET-N is a greedy algorithm, the results it outputs are *near-optimal*.

Lemma 6. Effectiveness of BASSET-N. *Let $\mathcal{I}$ be the subset of nodes selected by Alg. 5 and $|\mathcal{I}| = k_0$. Then, $g(s,t,\mathcal{I}) \geq (1-1/e)max_{|\mathcal{J}|=k_0}\ g(s,t,\mathcal{J})$, where $g(s,t,\mathcal{I})$, and $g(s,t,\mathcal{J})$ are defined by eq. (5.1).*

[3] This is because in random walk with restart, we have $\mathbf{r}(i,j) = \mathbf{Q}(j,i)$ for any i, j [TFP08].

Proof: It is easy to verify that the node i_0 selected in step 10 of Alg. 5 satisfies $i_0 = \text{argmax}_{j \notin \mathcal{I}, j \neq s, j \neq t} \text{g}(s, t, \mathcal{I} \bigcup j)$. Also, we have $\text{g}(s, t, \phi) = 0$, where ϕ is an empty set. On the other hand, according to Theorem 2, $\text{g}(s, t, \mathcal{I})$ is sub-modular wrt the subset $\mathcal{I}$. Therefore, we have $\text{g}(s, t, \mathcal{I}) \geq (1 - 1/e)\text{max}_{|\mathcal{J}|=k_0}\ \text{g}(s, t, \mathcal{J})$, which completes the proof. □

Next, we analyze the efficiency of BASSET-N, which is given in Lemma 7[4]. We can draw the following two conclusions, according to Lemma 7: (1) the proposed BASSET-N achieves a significant speedup over the straight-forward method ($O(n \cdot k^4)$ vs. $O(\binom{n}{k} m)$). For example, in the graph with 100 nodes and 1,000 edges, in order to find the gateway with $k = 5$ nodes, BASSET-N is more than *6 orders of magnitude* faster, and the speedup quickly increases wrt the size of the graph; (2) the proposed BASSET-N is applicable to large graphs since it is linear wrt the number of the nodes.

Lemma 7. Efficiency of BASSET-N. *The computational complexity of Alg. 5 is upper bounded by* $O(n \cdot k^4)$.

Proof: The cost for steps 1-2 is constant. The cost for step 3 is $O(n)$. At each inner loop (steps 6-7), the cost is $O(nj^3 + nj^2)$. The cost for steps 9-13 is $O(n)$. The outer loop has no more than $k - 1$ iterations. Putting these together, the computational cost for BASSET-N is:

$$\begin{aligned} & \text{Cost(BASSET-N)} \leq n + \sum_{j=1}^{k} (nj^3 + nj^2 + n) \\ = \ & n + nk + n\frac{k(k+1)(2k+1)}{6} + n\frac{k^2(k+1)^2}{4} \\ = \ & O(nk^4) \end{aligned} \tag{5.18}$$

which completes the proof. □

5.4.3 BASSET-G for Problem 4

BASSET-G Algorithm

Our fast solution for Problem 4 is summarized in Alg. 6. It works in a similar way as Alg. 5: after initialization (step 1), we first pick a node i_0 with the highest $\sum_{s \in \mathcal{S},\ t \in \mathcal{T}} \frac{\mathbf{r}(s,i)\mathbf{r}(i,t)}{\mathbf{r}(i,i)}$ (step 3). Then, in steps 4-14, we find the rest of the nodes in a greedy way. That is, in each outer-loop, we try to find one more node while keeping the current $\mathcal{I}$ unchanged. If the current subset of the nodes $\mathcal{I}$ can completely disconnect the source group and the target group (by setting them as sinks), we will stop the algorithm (step 10). As in Alg. 5, all the proximity scores are computed from the original graph $\mathbf{A}$. Therefore, we can again utilize those powerful pre-computation based methods to accelerate the whole process.

[4]Here, we assume that the cost to get one proximity score is constant, which can be achieved with pre-computation methods (e.g., B_LIN in Chapter 2.

Algorithm 6 BASSET-G

Require: the normalized adjacency matrix $\mathbf{A}$, the source group $\mathcal{S}$, the target group $\mathcal{T}$, the budget k and the parameter c

Ensure: a set of nodes $\mathcal{I}$, where $|\mathcal{I}| \leq k$.

1: initialize $\mathcal{I}$ to be empty.
2: compute the proximity score $\sum_{s\in\mathcal{S},\ t\in\mathcal{T}} \mathbf{r}(s,t)$ from the source group $\mathcal{S}$ to the target group $\mathcal{T}$.
3: find $i_0 = \text{argmax}_i \sum_{s\in\mathcal{S},\ t\in\mathcal{T}} \frac{\mathbf{r}(s,i)\mathbf{r}(i,t)}{\mathbf{r}(i,i)}$, where $i = 1, ..., n$ and $i \neq s, i \neq t$; add i_0 to $\mathcal{I}$.
4: **for** $j = 2$ to k **do**
5: **for** $i = 1$ to n, and $i \neq s, i \neq t$ and $i \notin \mathcal{I}$ **do**
6: let $\mathcal{J} = \mathcal{I} \cup i$.
7: compute $\mathbf{v}(i)$ as $\mathbf{v}(i) = \sum_{s\in\mathcal{S},\ t\in\mathcal{T}} \mathbf{r}(\mathcal{J},t)'\mathbf{r}(\mathcal{J},\mathcal{J})^{-1}\mathbf{r}(s,\mathcal{J})'$
8: **end for**
9: **if** $\max_i \mathbf{v}(i) \leq \sum_{s\in\mathcal{S},\ t\in\mathcal{T}} \mathbf{r}(s,t)$ **then**
10: find $i_0 = \text{argmax}_i \mathbf{v}(i)$; add i_0 to $\mathcal{I}$.
11: **else**
12: break;
13: **end if**
14: **end for**
15: return $\mathcal{I}$

Analysis of BASSET-G.

The effectiveness and efficiency of the proposed BASSET-G are given in Lemma 8 and Lemma 9, respectively. Similar as BASSET-N, the proposed BASSET-G is (1) *near-optimal*; and (2) fast and scalable for large graphs.

Lemma 8. Effectiveness of BASSET-G. *Let $\mathcal{I}$ be the subset of nodes selected by Alg. 6 and $|\mathcal{I}| = k_0$. Then, $g(\mathcal{S},\mathcal{T},\mathcal{I}) \geq (1-1/e)max_{|\mathcal{J}|=k_0}\ g(\mathcal{S},\mathcal{T},\mathcal{J})$, where $g(\mathcal{S},\mathcal{T},\mathcal{I})$, and $g(\mathcal{S},\mathcal{T},\mathcal{J})$ are defined by eq. (5.9).*

Proof: It is easy to verify that the node i_0 selected in step 10 of Alg. 6 satisfies $i_0 = \text{argmax}_{j\notin\mathcal{I}, j\notin\mathcal{S}, j\notin\mathcal{T}} \text{g}(\mathcal{S},\mathcal{T},\mathcal{I}\bigcup j)$. Also, we have that $\text{g}(\mathcal{S},\mathcal{T},\phi) = 0$, where ϕ is an empty set. On the other hand, according to Theorem 2, $\text{g}(\mathcal{S},\mathcal{T},\mathcal{I})$ is sub-modular wrt the subset $\mathcal{I}$. Therefore, we have $\text{g}(\mathcal{S},\mathcal{T},\mathcal{I}) \geq (1-1/e)\text{max}_{|\mathcal{J}|=k_0}\ \text{g}(\mathcal{S},\mathcal{T},\mathcal{J})$, which completes the proof. □

Lemma 9. Efficiency of BASSET-G. *The computational complexity of Alg. 6 is upper bounded by $O(n\cdot(max(k,|\mathcal{S}|,|\mathcal{T}|))^4)$.*

Proof: The cost for steps 1-2 is constant. The cost for step 3 is $O(n|\mathcal{S}||\mathcal{T}|)$. At each inner loop (steps 6-7), the cost is $O(n|\mathcal{S}||\mathcal{T}| + nj^3 + n|\mathcal{S}||\mathcal{T}|j^2)$. The cost for steps 9-13 is $O(n)$. The outer loop has no more than $k-1$ iterations. Putting these together, the computational cost for BASSET-N is:

$$\begin{aligned} &\text{Cost(BASSET-G)} \leq n|\mathcal{S}||\mathcal{T}| + \sum_{j=1}^{k}(n|\mathcal{S}||\mathcal{T}| + nj^3 + n|\mathcal{S}||\mathcal{T}|j^2 + n) \\ = \ & O(n(\text{max}(k,|\mathcal{S}|,|\mathcal{T}|))^4) \end{aligned} \tag{5.19}$$

which completes the proof. □

5.5 Experimental Evaluations

In this section we present experimental results. All the experiments are designed to answer the following questions:

- *Effectiveness:* how effective are the proposed 'Gateway-ness' scores in real graphs?
- *Efficiency:* how fast and scalable are the proposed BASSET-N and BASSET-G?

5.5.1 Experimental Setup

Data sets. We used five real data sets, which are summarized in table 7.2.

Table 5.2: Summary of the data sets

Name	n	m	**Weight**
Karate	34	152	No
PolBooks	105	882	No
AC	421,807	2,133,632	No
AA	418,236	2,753,798	Yes
NetFlix	2,667,199	56,919,190	No

The first data set (*Karate*) is an un-weighted unipartite graph, which describes friendship among the 34 members of a karate club at a US university [Zac77]. Each node is a member in the karate club and the existence of the edge indicates that the two corresponding members are friends. Overall, we have $n = 34$ nodes and $m = 156$ edges.

The second data set (*PolBooks*) is a co-purchasing book network.[5] Each node is a political book and there is an edge between two books if purchased by the same person. Overall, we have $n = 105$ nodes and $m = 882$ edges.

The third data set (*AC*) and the fourth data set (*AA*) are both from DBLP.[6] The third data set (*AC*) is an un-weighted bipartite graph. We have two types of nodes: author and conference. The existence of the edge indicates that the corresponding author has published in the corresponding conference. Overall, we have $421,807$ nodes and $m = 2,667,199$ edges.

The fourth data set (*AA*) is a co-authorship network, where each node is an author and the edge weight is the number of the co-authored papers between the two corresponding persons. Overall, we have $n = 418,236$ nodes and $m = 2,753,798$ edges.

The last data set (*NetFlix*) is from the Netflix prize[7]. Rows represent users and columns represent movies. If a user has given a particular movie positive ratings (4 or 5), we connect them with

[5] `http://www.orgnet.com/`
[6] `http://www.informatik.uni-trier.de/~ley/db/`
[7] `http://www.netflixprize.com/`

an edge. In total, we have 2,667,199 nodes (2,649,429 users and 17,770 movies), and 56,919,190 edges.

Parameter settings and machine configurations. There is one parameter in BASSET-N and BASSET-G, the probability c for random walk with restart. We set $c = 0.95$, as suggested in [TFP08]. For the computational cost, we report the wall-clock time. All the experiments ran on the same machine with four 2.4GHz AMD CPUs and 48GB memory, running Linux (2.6 kernel). For each experiment, we run it 10 times and report the average.

5.5.2 Effectiveness

Here, we evaluate the effectiveness of the proposed 'Gateway-ness' scores. We first compare with several candidate methods in terms of separating the source from the target. And then, we present various case studies.

Quantitative Comparisons

The basic idea of the proposed 'Gateway-ness' scores is to find a subset of nodes which collectively play an important role in measuring the proximity from the source node (or source group) to the target node (or target group). Here, we want to validate this basic assumption. We compare it with the following alternative choices: (a) selecting k nodes with the highest center-piece AND score (CePS-AND) [TF06]; (b) selecting k nodes with the highest center-piece OR score (CePS-OR) [TF06]; (c) randomly selecting k nodes (Rand); (d) randomly selecting k nodes from the neighboring nodes of the source node and the target node (Neighbor-Rand); (e) selecting k nodes with the highest $\frac{\mathbf{r}(s,i)\mathbf{r}(i,t)}{\mathbf{r}(i,i)}$(Topk-Ind). We randomly select a source node s and a target node t,[8] and then use the different methods to select a subset $\mathcal{I}$ with k nodes. Figure 2 presents the comparison results, where the x-axis is the number of nodes selected (k), and the y-axis is the normalized decay in terms of the proximity score from the source node s to the target node t ($\frac{\mathbf{r}(s,t)-\mathbf{r}_{\mathcal{I}}(s,t)}{\mathbf{r}(s,t)}$). The resulting curves are averaged over 1,000 randomly chosen source-target pairs. From figure 5.2, we can see that (1) the proposed BASSET-N performs best in terms of separating the source from the target; (2)Topk-Ind, where we simply select k nodes with highest $\frac{\mathbf{r}(s,i)\mathbf{r}(i,t)}{\mathbf{r}(i,i)}$, does not perform as well as BASSET-N, where we want to find a subset of k nodes which *collectively* has the highest score $\mathbf{r}(\mathcal{I},t)'\mathbf{r}(\mathcal{I},\mathcal{I})^{-1}\mathbf{r}(s,\mathcal{I})'$.

Case Studies

Next, we will show some case studies, to demonstrate the effectiveness of BASSET-N and BASSET-G.

Karate. We start with *Karate* graph, which is widely used in social network analysis. In figure 7.11, there are two different communities in the graph (shaded). In each community, there are some 'hub' nodes (e.g., nodes 33 and 34 in the left community; and nodes 1 and 4 in the right community). The two communities are connected by some 'bridging nodes' (e.g., nodes 3, 10,

[8]The result when source and target are a group of nodes is similar, and omitted for brevity.

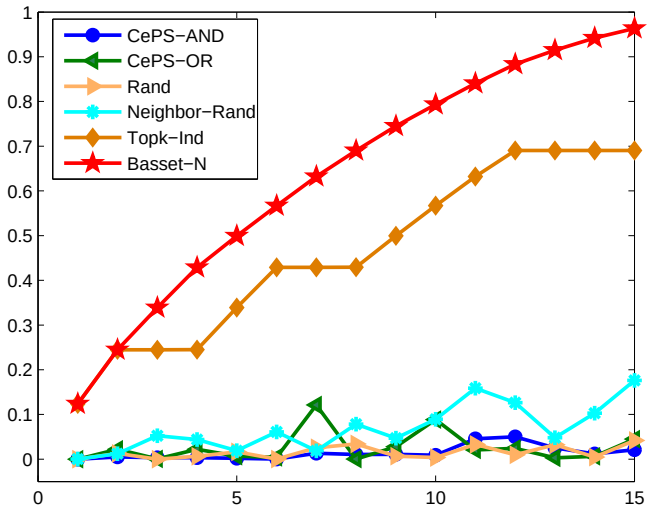


Figure 5.2: Effectiveness comparison between BASSET-N and alternatives. Normalized decay of proximity vs. k. Higher is better. The proposed BASSET-N (red star) is the best.

19, 20). Table 5.3 presents the resulting gateways of BASSET-N with the budget $k = 5$ for a few source-target pairs. The results are consistent with human intuition. The gateways either are the local center of the community that the source/target node belongs to, or are bridging nodes that connect the two communities when the source node and the target node belong to different communities. For example, if $s = 1$ and $t = 33$, the resulting nodes 3, 10, 11 are bridging nodes, while node 34 is the local center for the left community. Note that, we always return less than or equal to $k = 5$ nodes. For example, if $s = 15$ and $t = 34$, we only output one node (node 1) as the gateway. This is because all the paths from node 15 to node 34 must go through node 1.

Table 5.3: BASSET-N on *Karate* graph

Source (s)	**Target (t)**	**Gateways ($\mathcal{I}$)**
24	31	{33,34}
15	34	{1}
1	33	{3,10,11,21,34}

PolBooks. For this data set, the nodes are political books and the existence of the edge indicates the co-purchasing (by the same person) of the two books. Each book is annotated by one of the following three labels: 'liberal', 'conservative' and 'neutral'. We pick a 'liberal' book ('The Price of Loyalty') as the source node, and a 'conservative' book ('Losing Bin Laden') as the target node. Then, we ran the proposed BASSET-N to find the gateway with 10 nodes. The result is presented in table 5.4. The result is again consistent with human intuition, - the resulting gateway books are either popular books in one of the two communities ('conservative' vs. 'liberal') such as, 'Bush

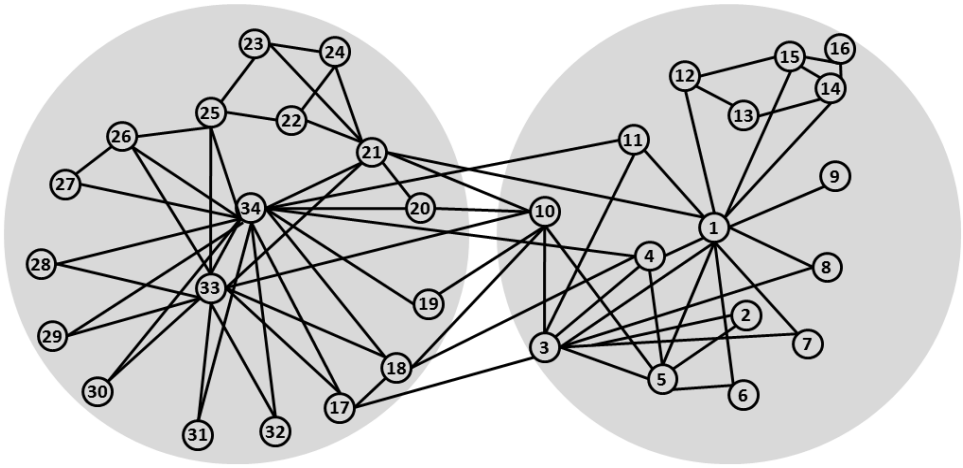


Figure 5.3: Karate graph

country' from 'conservative', 'Back up suck up' from 'liberal', etc; or those 'neutral' books which are likely to be purchased by readers from both communities (e.g., 'Sleeping with the devil', etc).

Table 5.4: BASSET-N on *PolBooks* Graph. ('c' for 'conservative', 'l' for 'liberal', and 'n' for 'neutral')

Node Index	**Book Title**	**Label**
10	Bush country	c
13	Off with their heads	c
103	Back up such up	l
5	Sleeping with the devil	n
8	Ghost wars	n
77	Plan of attack	n
78	Bush at war	c
59	Rise of the vulcanes	c
52	Allies	c
42	The Bushes	c

AC. This is a bipartite graph. Given a source conference/author and a target conference/author, we can run BASSET-N to find either the gateway conferences or the gateway authors. Table 5.5 gives one such example when the source is 'VLDB' and the target is 'NIPS'. Conceptually, we treat an $n_1 \times n_2$ bipartite graph as a $(n_1 + n_2) \times (n_1 + n_2)$ unipartite graph, and we further restrict the search to the desired node type. Again, we can see that the results make sense. The resulting gateway authors are either productive in one of the two fields: databases vs. statistics, (e.g., Prof. Michael I. Jordan in statistics, Prof. Hector Garcia-Molina in databases, etc); or productive in data mining (e.g., Dr. Rakesh Agrawal, Prof. Jiawei-Han), which is an intersection field between statistics and databases. We have similar observations for the resulting gateway conferences. For example, 'SIGMOD' and 'UAI' are isomorphic (i.e., have very similar neighbor sets) to 'VLDB'

and ‘NIPS’, respectively; and ‘KDD’ is one major conference in data mining, which is a highly plausible major connection from ‘VLDB’ (databases) to ‘NIPS’ (statistics / machine learning).

Table 5.5: BASSET-N on *AC* graph. From the source ‘VLDB’ to the target ‘NIPS’.

Gateway Authors	Michael I. Jordan,Philip S. Yu, Jiawei Han,Geoffrey E. Hinton, H. V. Jagadish,Christos Faloutsos, Sebastian Thrun,Rakesh Agrawal, Hector Garcia-Molina,Raghu Ramakrishnan
Gateway Conferences	SIGMOD,ICDE,ICML, IJCAI,KDD, AAAI,CIKM,ICANN,SAC,UAI

Table 5.6: BASSET-G on *AA* Network.

Source Group	**Target Group**	**Gateway** (k=*10*)
Tom M. Mitchell, William W. Cohen, Jaime G. Carbonell	David J. Dewitt, Hector Garcia-Molina, H. V. Jagadish	Sunita Sarawagi, Christos Faloutsos, James P. Callan, Yiming Yang, Rakesh Agrawal, Andrew Y. Ng, Rich Caruana, Andrew Mccallum, Chengxiang Zhai, Sebastian Thrun,

(a) A group of people in ‘text’ to a group of people in ‘databases’

Source Group	**Target Group**	**Gateway** (k=*10*)
Manuel Blum, Christos H. Papadimitriou	Vipin Kumar, Wei Wang, George Karypis, Mohammed J. Zaki	Philip S. Yu, Mihalis Yannakakis, Hui Xiong, , Prabhakar Raghavan, Sally A. Goldman, Jiawei Han, Moni Naor, Mitsunori Ogihara, Richard M. Karp, Hongjun Lu

(b) A group of people in ‘theory ’ to a group of people in ‘bioinfomatics’

AA. We use this data set to perform case studies for the proposed BASSET-G. We choose (1) a group of people from a certain field (e.g., ‘text’, ‘theory’, etc) as the source group $\mathcal{S}$; and (2) another group of people in some other field (e.g., ‘databases’, ‘bioinfomatics’, etc) as the target group $\mathcal{T}$. Then, we ran the proposed BASSET-N to find the gateway with $k = 10$ nodes. Table 5.6 lists some results. They are all consistent with human intuition, - the resulting authors are either productive authors in one of the two fields, or multi-disciplinary, who have close collaborations to both the source and the target groups of authors.

5.5.3 Efficiency

We will study the wall-clock running time of the proposed BASSET-N and BASSET-G here. Basically, we want to answer the following two questions:

1. *(Speed)* What is the speedup of the proposed BASSET-N and BASSET-G over the straightforward methods?
2. *(Scalability)* How do BASSET-N and BASSET-G scale with the size of the graph (n and m)?

First, we compare BASSET-N and BASSET-G with two straightforward methods: (1) 'Com-RWR', where we use combinatorial enumeration to find the gateway and, for each enumeration, we compute the proximity from the *new* graph; and (2) 'Com-Eval', where we use combinatorial enumeration to find the gateway, and for each enumeration, we compute the proximity from the *original* graph. Figure 5.4 and figure 5.5 show the comparison on two real data sets. We can draw the following conclusions. (1) Straightforward methods ('Com-RWR' and 'Com-Eval') are computationally intractable even for a small graph. For example, on the *Karate* data set with only 34 nodes, it takes more than $20,560$ seconds and $100,000$ seconds to find the $k = 10$ gateway by 'Com-Eval' and by 'Com-RWR', respectively. (2) The speedup of the proposed BASSET-N and BASSET-G over both 'Com-Eval' and 'Com-RWR' is significant - in most cases, we achieve *several (up to 6) orders of magnitude* speedups. (3) The speedup of the proposed BASSET-N and BASSET-G over both 'Com-RWR' and 'Com-Eval' quickly increases wrt the size of the gateway k. Note that we stop running the program if it takes more than 100,000 seconds (i.e., longer than a day).

Next, we evaluate the scalability of the proposed BASSET-N and BASSET-G wrt the size of the graph, using the largest data set (*NetFlix*). From figure 5.6 and figure 5.7, we can make the following conclusions: (1) if we fix the number of nodes (n) in the graph, the wall-clock time of both BASSET-N and BASSET-G is almost *constant* wrt the number of edges (m); and (2) if we fix the number of edges (m) in the graph, the wall-clock time of both BASSET-N and BASSET-G is *linear* wrt the number of nodes (n). Therefore, they are suitable for large graphs.

5.6 Related Work

In this section, we review the related work, which can be categorized into two parts:

Betweenness centrality. The proposed 'Gateway-ness' scores relate to measures of betweenness centrality, both those based on the shortest path [Fre77], as well as those based on random walk [New05]. When the gateway set size is $k = 1$, the proposed 'Gateway-ness' scores can be viewed as *query-specific* betweenness centrality measures. Moreover, in the proposed BASSET-N and BASSET-G, we aim to find a subset of nodes *collectively*, wherein traditional betweenness centrality, we usually calculate the score for each node *independently* (and then might pick k nodes with the highest individual scores).

Connection subgraphs. In the proposed BASSET-N, the idea of finding a subset of nodes wrt the source/target is also related to the concept of connection subgraphs, such as [FMT04, KNV06, TF06]. However, in connection subgraphs, we aim to find a subset of nodes which have *strong* connections among themselves for the purpose of visualization. While in the proposed BASSET-

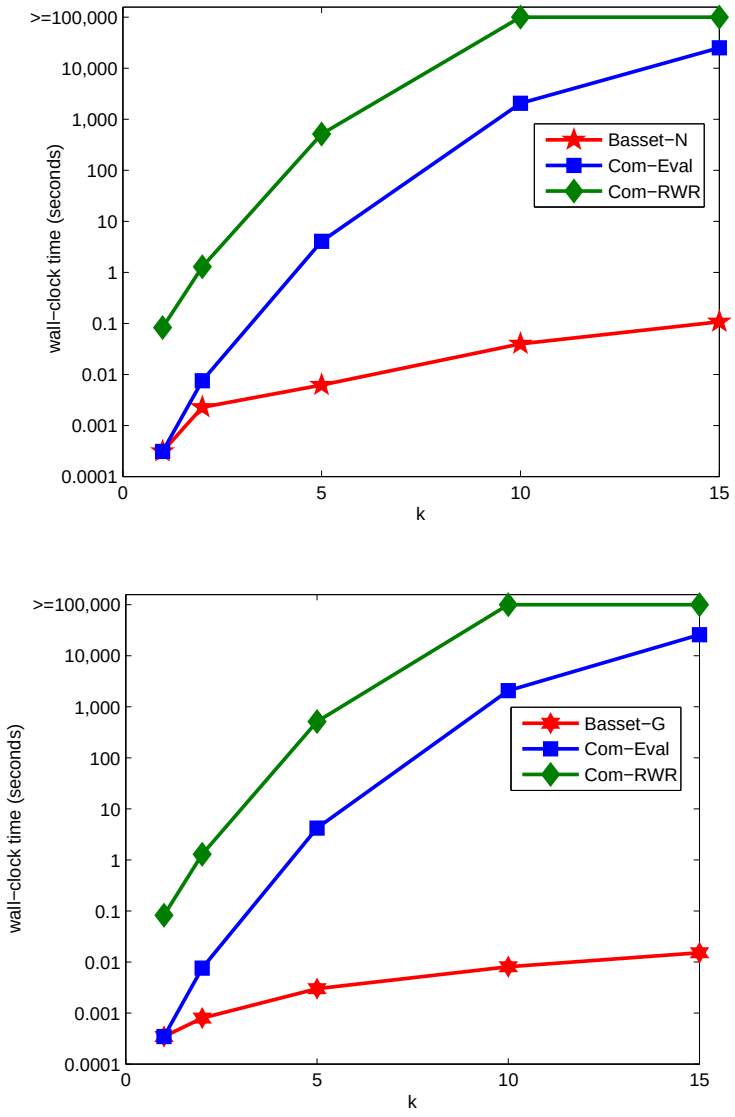


Figure 5.4: Comparison of speed on *Karate* graph. Wall-clock time vs. k. Lower is better. Time is in logarithm scale. The proposed BASSET-N and BASSET-G (red star) are significantly faster.

N, we implicity encourage the resulting subset of nodes to be disconnected with each other so that they are able to *collectively disconnect* the target node from the source node to the largest extent (if we set them as sinks). It is interesting to notice that, if we want to find the gateway with $k = 1$ for BASSET-N, it can be viewed as a *normalized directed* version of CePS-AND score [TF06].[9] Moreover, We allow the more general case where the source/target is a group of nodes in the

[9]To see this, notice that in the case $k = 1$, in BASSET-N, we want to find the node with the highest $\frac{\mathbf{r}(s,i)\mathbf{r}(i,t)}{\mathbf{r}(i,i)}$; while in CePS-AND [TF06], it picks the nodes with the highest $\mathbf{r}(s,i)\mathbf{r}(t,i)$, where $i = 1, ..., n$ and $i \neq s, i \neq t$.

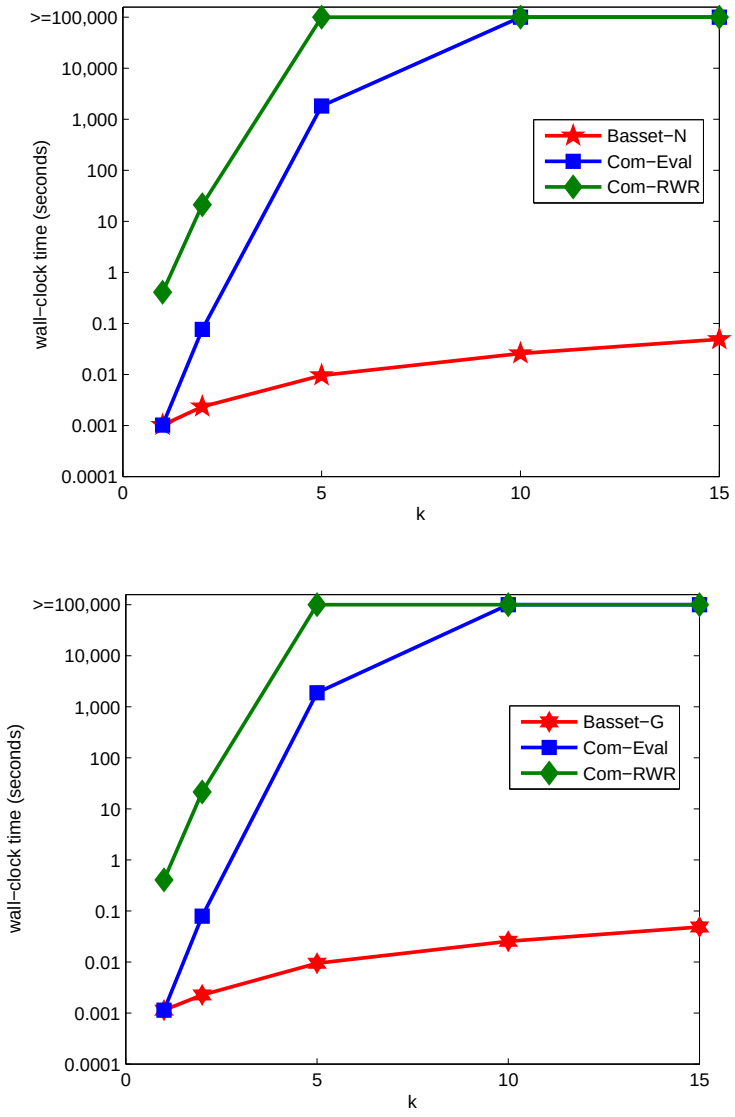
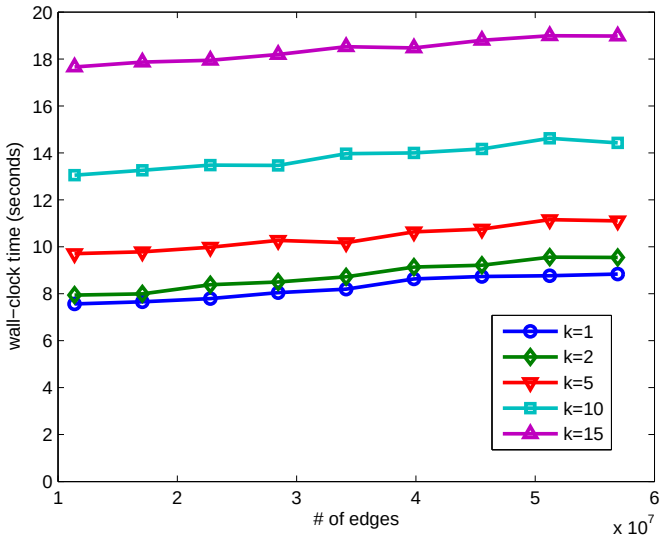


Figure 5.5: Comparison of speed on *PolBooks* graph. Wall-clock time vs. k. Lower is better. Time is in logarithm scale. The proposed BASSET-N and BASSET-G (red star) are significantly faster.

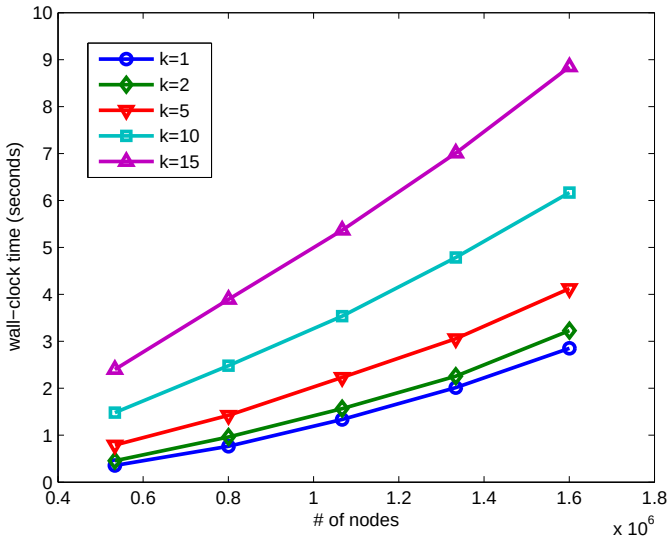
proposed BASSET-G; however in connection subgraphs, the source/target is always a single node.

5.7 Conclusion

In this chapter, we study how to find good 'gateway' nodes in a graph, given one or more source and target nodes. Our main contributions are: (a) we formulate the problem precisely; (b) we develop BASSET-N and BASSET-G, two fast (up to *6,000,000x* speedup) and scalable (*linear* wrt



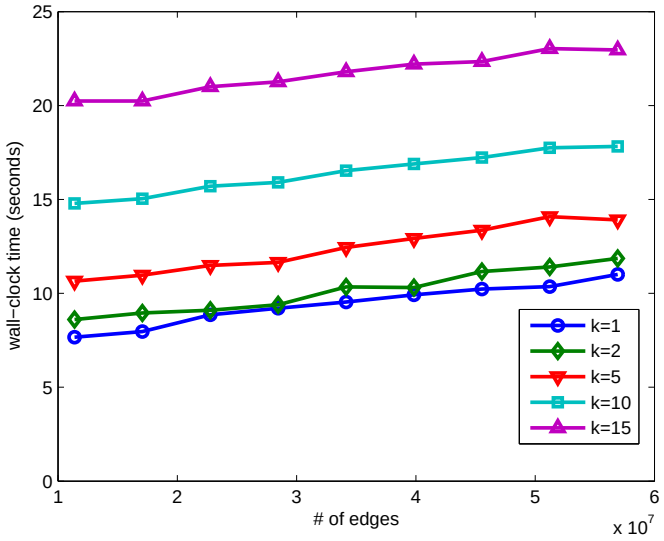
wall-clock time vs. m
(fix $n = 2,667,199$)



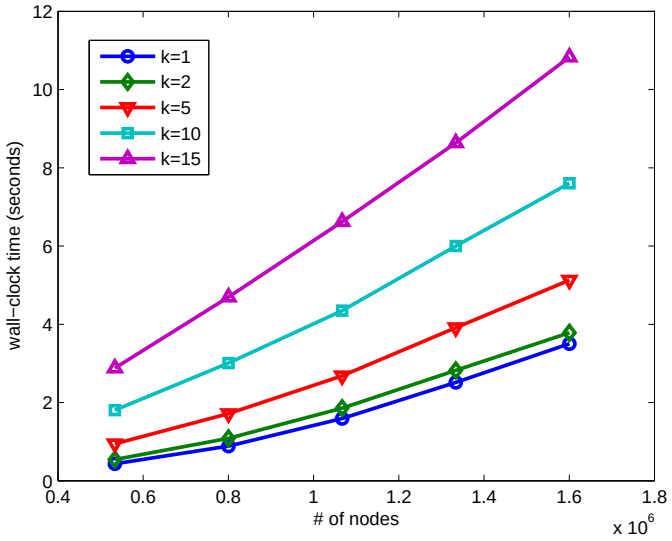
wall-clock time vs. n
(fix $m = 23,874,700$)

Figure 5.6: Scalability of BASSET. Wall-clock time vs. the size of the graph. Lower is better. $|\mathcal{S}| = |\mathcal{T}| = 5$.

the number of the nodes in the graph) algorithms to solve it in a provably near-optimal fashion, using sub-modularity. We applied the proposed BASSET-N and BASSET-G on real data sets to validate the effectiveness and efficiency.



wall-clock time vs. m
(fix $n = 2,667,199$)



wall-clock time vs. n
(fix $m = 23,874,700$)

Figure 5.7: Scalability of BASSET-G. Wall-clock time vs. the size of the graph. Lower is better. $|\mathcal{S}| = |\mathcal{T}| = 5$.

Part III

Querying Dynamic Graphs

Chapter 6

Proximity Tracking

Summary of This Chapter

- **Questions we want to answer:**

  Q1: How to define a good proximity score in a dynamic setting (i.e., graphs are changing over time)?

  Q2: How to incrementally track the proximity between nodes of interest, as edge are updated?

- **Our answers and contributions**

  A1: We proposed a novel proximity and centrality score for time-evolving graphs.

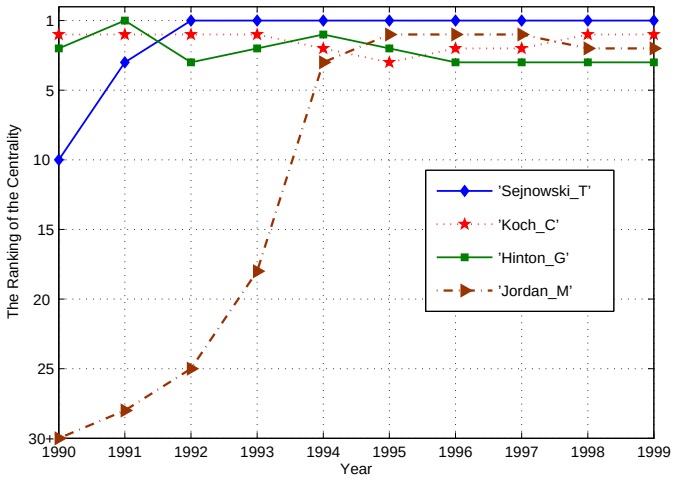
  A2: We proposed two fast incremental algorithms, achieving *176x* speedup, without quality loss.

6.1 Introduction

Measuring proximity (a.k.a relevance) between nodes on bipartite graphs (see [Koz92] 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 and many more.

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. Consider an author-conference evolving graph, which effectively contains information about the number of papers (edge weights) published by each author (type 1 node) in each conference (type 2 node) for each year (timestamp). Trend analysis tools are becoming very popular. For example, Google Trends[1] provides useful insights, despite the simplicity of its approach. For instance, in the setting of our example, a tool similar to Google Trends might answer questions such as "*How does*

[1] `http://www.google.com/trends/`



(a) The ranking of centrality for some authors in NIPS.

ICDE	CIKM	KDD	ICDM
ICDCS	ICDCS	SIGMOD	KDD
SIGMETRICS	ICDE	ICDM	ICDE
PDIS	SIGMETRICS	CIKM	SDM
VLDB	ICMCS	ICDCS	VLDB
1992	1997	2002	2007

(b) Philip S. Yu's top 5 conferences at four time steps, using a window of 5 years.

Figure 6.1: Scaling sophisticated trend analysis to time-evolving graphs. See Section 6.6.3 for detailed description of results.

the number of papers published by an author vary over time?" or "*How does the number of papers published in a particular conference or research area (i.e., set of conferences) vary over time?*" This kind of analysis takes into account paper counts for either an author or a conference alone or, at best, a single, specific author-conference pair. Instead, we want to employ powerful analysis tools inspired by the well-established model of random walk with restart to analyze the entire graph and provide further insight, taking into account all author-conference information so far, i.e., including indirect relationships among them. However, if we need to essentially incorporate all pairwise relationships in the analysis, scalability quickly becomes a major issue. This is precisely the problem we address in this chapter: how can we efficiently keep track of proximity and avoid global re-computation as new information arrives. Fig. 6.1 shows examples of our approach.

In this chapter, we address such challenges in multiple dimensions. In particular, this chapter addresses 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?

We begin in Section 2 with the problem definition and, in Section 3, we propose our proximity definition for dynamic bipartite graphs. We carefully design our measurements to deal with (1) the links arriving at different time steps and (2) important properties, such as monotonicity. Proximity will also serve as the basis of our centrality measurement in the dynamic setting. Then, in Section 4, 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 5. In Section 6, we verify the effectiveness and efficiency of our proposed dynamic proximity on real datasets.

The major contributions of this chapter can be summarized as follows:

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 any-time fashion.

4: Extensive experimental case-studies on several real datasets, showing how different queries can be answered, achieving up to **15∼176x** speed-up.

6.2 Problem Definitions

Table 6.1 lists the main symbols we use throughout the paper. 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 i^{th} row and j^{th} column of the matrix $\mathbf{M}$, and $\mathbf{M}(i, :)$ is the i^{th} row of $\mathbf{M}$, etc. Without loss of generality, we assume that the numbers of type 1 and type 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 paper, 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

Table 6.1: 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

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* dataset (see Section 6), although we have more than 400,000 authors and about 2 million edges, with only ~ 3500 conferences. In the user-movie graph from the *NetFlix* dataset, 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 5. pTrack*(Proximity Tracking)*

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 5 to deal with *pTrack*.

Problem 6. cTrack*(Centrality Tracking)*

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 (i) of *cTrack*, we do not need the query nodes as inputs. We will propose another algorithm (*Track-Centrality*) in Section 5 to deal with *cTrack*.

For all these tasks (*pTrack* and *cTrack*), we want to provide any-time answers. That is, we want to quickly maintain up-to-date answers as soon as we observe a new slice matrix $\mathbf{S}^{(t)}$. Some representative examples of our methods are also shown in Figure 6.1.

6.3 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.

6.3.1 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.

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 [PYFD04]. 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 uni-partite graph with the following square adjacency matrix $\mathbf{W}$ and degree matrix $\mathbf{D}$:

$$\mathbf{W} = \begin{pmatrix} \mathbf{0}_{n\times n} & \mathbf{M} \\ \mathbf{M}' & \mathbf{0}_{l\times l} \end{pmatrix}$$
$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_1 & \mathbf{0}_{n\times l} \\ \mathbf{0}_{l\times n} & \mathbf{D}_2 \end{pmatrix} \tag{6.1}$$

then, all the proximity scores $r_{i,j}$ between all possible node pairs i, j are determined by the matrix $\mathbf{Q}$:

$$\begin{aligned} r_{i,j} &= \mathbf{Q}(i,j) \\ \mathbf{Q} &= (1-c)\cdot(\mathbf{I}_{(n+l)\times(n+l)} - c\mathbf{D}^{-1}\mathbf{W})^{-1} \end{aligned} \tag{6.2}$$

Based on the dynamic proximity as in equation 6.4, 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,

$$\text{centrality}(s) \triangleq \frac{\sum_{i=1}^{n+l} r_{i,s}}{n+l} \tag{6.3}$$

6.3.2 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 equation 6.2) 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 subsection 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. In subsection 3.2.2, we propose *degree-preservation* to achieve this property.

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 $\beta^j (\beta > 1)$: $\mathbf{M}^{(t)} = \sum_{j=1}^{t} \beta^j \mathbf{S}^{(j)}$.

Fixed degree matrix.

In a dynamic setting, if we apply the actual degree matrix $\mathbf{D}^{(t)}$ to equation (6.2) at time t, the monotonicity property will not hold. To address this issue, we propose to use degree-preservation [KNV06, TKF07]. 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 equation (6.4). The adjacency matrix $\mathbf{M}^{(t)}$ is computed by any update method in subsection 3.2 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.

$$\begin{aligned} r_{i,j}^{(t)} &= \mathbf{Q}^{(t)}(i,j) \\ \mathbf{Q}^{(t)} &= (1-c)\cdot(\mathbf{I}_{(n+l)\times(n+l)} - c\tilde{\mathbf{D}}^{-1}\mathbf{W}^{(t)})^{-1} \\ \mathbf{W}^{(t)} &= \begin{pmatrix} \mathbf{0}_{n\times n} & \mathbf{M}^{(t)} \\ \mathbf{M}'^{(t)} & \mathbf{0}_{l\times l} \end{pmatrix} \\ \tilde{\mathbf{D}} &= a\cdot\mathbf{D}^{(1)} \end{aligned} \tag{6.4}$$

We have the following lemma for our dynamic proximity (equation (6.4)). By the lemma 10, 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 10. 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 $\mathbf{Q}^{(t)} = (1-c)\sum_{k=0}^{\infty}(c\tilde{\mathbf{D}}^{-1}\mathbf{W}^{(t)})^k$. 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 are not affected. If non-monotonic scores are permissible, none of these assumptions are necessary.

6.4 Dynamic Proximity: Computations

6.4.1 Preliminaries: BB_LIN on Static Graphs

In this section, we introduce our fast solutions to efficiently track dynamic proximity.

One problem with random walk with restart is computational efficiency, especially for large graphs. According to the definition (equation (6.4)), we need to invert an $(n+l)\times(n+l)$ matrix. This operation is prohibitively slow for large graphs. In Chapter 2, we proposed BB_LIN for skewed, static bipartite graphs, with which we only need to pre-compute and store a matrix inversion of size $l \times l$ to get all possible proximity scores.

Based on BB_LIN, we only need to pre-compute and store a matrix inversion $\mathbf{\Lambda}$ of size $l \times l$. For skewed bipartite graphs ($l \ll m, n$), $\mathbf{\Lambda}$ is much cheaper to pre-compute and store. For example, on the entire *NetFlix* user-movie bipartite graph, which contains about $2.7M$ users, about $18K$ movies and more than $100M$ edges (see Section 6 for the detailed description of the data set), it takes 1.5

Algorithm 7 GetQij

Require: The core matrix $\mathbf{\Lambda}$, the normalized adjacency matrices $\mathbf{Mr}$ (for type 1 objects), and $\mathbf{Mc}$ (for type 2), and the query nodes i and j $(1 \leq i, j \leq (n+l))$.

Ensure: The proximity $r_{i,j}$ from node i to node j

1: **if** $i \leq n$ and $j \leq n$ **then**
2: $\quad q(i,j) = 1(i=j) + c^2\mathbf{Mr}(i,:) \cdot \mathbf{\Lambda} \cdot \mathbf{Mc}(:,j)$
3: **else if** $i \leq n$ and $j > n$ **then**
4: $\quad q(i,j) = c\mathbf{Mr}(i,:) \cdot \mathbf{\Lambda}(:, j-n)$
5: **else if** $i > n$ and $j \leq n$ **then**
6: $\quad q(i,j) = c\mathbf{\Lambda}(i-n,:) \cdot \mathbf{Mc}(:,j)$
7: **else**
8: $\quad q(i,j) = \mathbf{\Lambda}(i-n, j-n)$
9: **end if**
10: **Return:** $r_{i,j} = (1-c)q(i,j)$

hours to pre-compute the $18K \times 18K$ matrix inversion $\mathbf{\Lambda}$. For pre-computation stage, this is quite acceptable.

On the other hand, in the on-line query stage, we can get any proximity scores using the function **GetQij**[2] . 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 (step 4 and step 6) is necessary. Finally, for a proximity score between two type-2 objects (step 8), only retrieving one element in the matrix $\mathbf{\Lambda}$ is needed. As an example, on the *NetFlix* dataset, it takes less than 1 second to get one proximity score. Note that all possible proximity scores are determined by the matrix $\mathbf{\Lambda}$ (together with the normalized adjacency matrices $\mathbf{Mr}$ and $\mathbf{Mc}$). We thus refer to the matrix $\mathbf{\Lambda}$ as the the *core matrix*.

6.4.2 Challenges for Dynamic Setting

In a dynamic setting, since the adjacency matrix changes over time, the core matrix $\mathbf{\Lambda}^{(t)}$ is no longer constant. In other words, the steps 1-4 in BB_LIN themselves become a part of the on-line stage since we need to update the core matrix $\mathbf{\Lambda}^{(t)}$ at each time step. If we still rely on the straightforward strategy (i.e., the steps 1-4 in BB_LIN 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 hours to recompute the core matrix for the *NetFlix* dataset is unacceptably long.

Thus, our goal is to efficiently update the core matrix $\mathbf{\Lambda}^{(t)}$ at time step t, based on the previous core matrix $\mathbf{\Lambda}^{(t-1)}$ and the difference matrix $\Delta\mathbf{M}^{(t)}$. For simplicity, we shall henceforth assume the use of the global aggregation scheme to update the adjacency matrix. However, the ideas can

[2]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.

be easily applied to the other schemes, sliding window and exponential weighting.

6.4.3 Our Solution 1: Single Update

Next, we describe a fast algorithm (*Fast-Single-Update*) to update the core matrix $\mathbf{\Lambda}^{(t)}$ 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: $[i_0, j_0, w_0]$.

Algorithm 8 *Fast-Single-Update*

Require: The core matrix $\mathbf{\Lambda}^{(t-1)}$, 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.

Ensure: The core matrix $\mathbf{\Lambda}^{(t)}$, 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: $\mathbf{L} = (\mathbf{I}_{2\times 2} - \mathbf{Y} \cdot \mathbf{\Lambda}^{(t-1)} \cdot \mathbf{X})^{-1}$
8: $\mathbf{\Lambda}^{(t)} = \mathbf{\Lambda}^{(t-1)} + \mathbf{\Lambda}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{\Lambda}^{(t-1)}$

The correctness of *Fast-Single-Update* is guaranteed by the following theorem:

Theorem 3. Correctness of *Fast-Single-Update*. *The matrix $\mathbf{\Lambda}^{(t)}$ maintained by* Fast-Single-Update *is exactly the core matrix at time step t, i.e., $\mathbf{\Lambda}^{(t)} = (\mathbf{I} - c^2\mathbf{Mc}^{(t)}\mathbf{Mr}^{(t)})^{-1}$.*

Proof: first of all, since only one edge (i_0, j_0) is updated at time t, only the i_0^{th} row of the matrix $\mathbf{Mr}^{(t)}$ and the i_0^{th} 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:

$$\begin{aligned} \mathbf{V}^t &= c^2 \sum_{k=1}^{n} \mathbf{Mc}^{(t)}(:, k) \cdot \mathbf{Mr}^{(t)}(k, :) \\ &= \mathbf{V}^{t-1} + \delta \end{aligned} \tag{6.5}$$

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 Alg. 8. Putting all the above together, we have

$$\mathbf{\Lambda}^t = (\mathbf{I} - \mathbf{V}^t)^{-1} = (\mathbf{I} - \mathbf{V}^{t-1} - \mathbf{X} \cdot \mathbf{Y})^{-1} \tag{6.6}$$

Applying the Sherman-Morrison Lemma [PC90] to equation (6.6), we have

$$\mathbf{\Lambda}^{(t)} = \mathbf{\Lambda}^{(t-1)} + \mathbf{\Lambda}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{\Lambda}^{(t-1)}$$

where the 2×2 matrix $\mathbf{L}$ is defined in step 7 of Alg. 8. This completes the proof. □

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 11. 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 2-3, $O(l)$ for steps 4-6 and $O(l^2)$ for steps 7-8. Putting it together, we have that the total cost for *Fast-Single-Update* is $O(l^2)$, which completes the proof. □

6.4.4 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 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 conferences 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:

$$\begin{aligned} \Delta\mathbf{M}r(k,s) &= \frac{\Delta\mathbf{M}^{(t)}(i_k, j_s)}{\tilde{\mathbf{D}}(i_k, i_k)} \\ \Delta\mathbf{M}c(s,k) &= \frac{\Delta\mathbf{M}^{(t)}(j_s, i_k)}{\tilde{\mathbf{D}}(j_s + n, j_s + n)} \\ (k = 1, ..., \hat{n}, s &= 1, ..., \hat{l}) \end{aligned} \tag{6.7}$$

The correctness of *Fast-Batch-Update* is guaranteed by the following theorem:

Algorithm 9 *Fast-Batch-Update*

Require: The core matrix $\mathbf{\Lambda}^{(t-1)}$, 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

Ensure: The core matrix $\mathbf{\Lambda}^{(t)}$, 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 (6.7)
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 $\hat{k} = \min(\hat{\mathrm{l}}, \hat{\mathrm{n}})$. let $\mathbf{X} = \mathbf{0}_{l \times 2\hat{k}}$, and $\mathbf{Y} = \mathbf{0}_{2\hat{k} \times l}$
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: $\mathbf{L} = (\mathbf{I}_{2\hat{k} \times 2\hat{k}} - \mathbf{Y} \cdot \mathbf{\Lambda}^{(t-1)} \cdot \mathbf{X})^{-1}$
23: $\mathbf{\Lambda}^{(t)} = \mathbf{\Lambda}^{(t-1)} + \mathbf{\Lambda}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{\Lambda}^{(t-1)}$

Theorem 4. Delta Matrix Inversion Theorem. *The matrix* $\mathbf{\Lambda}^{(t)}$ *maintained by* Fast-Batch-Update *is exactly the core matrix at time step* t*, i.e.,* $\mathbf{\Lambda}^{(t)} = (\mathbf{I} - c^2\mathbf{Mc}^{(t)}\mathbf{Mr}^{(t)})^{-1}$.

Proof: Let $\mathbf{V}^{(t)} = c^2\mathbf{Mc}^{(t)} \cdot \mathbf{Mr}^{(t)}$, and $\mathbf{V}^{(t-1)} = c^2\mathbf{Mc}^{(t-1)} \cdot \mathbf{Mr}^{(t-1)}$. Similar as the proof for theorem 3, we have

$$\mathbf{V}^{(t)} = \mathbf{V}^{(t-1)} - \mathbf{X} \cdot \mathbf{Y} \tag{6.8}$$

where the matrices $\mathbf{X}$ and $\mathbf{Y}$ are defined in steps 6-21 of Alg. 9.

Applying the Sherman-Morrison Lemma [PC90] to equation (6.8), we have

$$\mathbf{\Lambda}^{(t)} = \mathbf{\Lambda}^{(t-1)} + \mathbf{\Lambda}^{(t-1)} \cdot \mathbf{X} \cdot \mathbf{L} \cdot \mathbf{Y} \cdot \mathbf{\Lambda}^{(t-1)}$$

where the $2\hat{k} \times 2\hat{k}$ matrix $\mathbf{L}$ is defined in step 22 of Alg. 9. This completes the proof. □

The efficiency of *Fast-Single-Update* is given by the following lemma. Note that the linear term $O(\hat{m})$ comes from equation (6.7), since we need to scan the non-zero elements of the difference matrix $\Delta\mathbf{M}^{(t)}$. 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 $\min(\hat{\mathrm{l}}, \hat{\mathrm{n}}) \ll \hat{\mathrm{m}}$, *Fast-Batch-Update* is much more efficient in this case.

Lemma 12. 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 11. Note that the linear term $O(\hat{m})$ comes from equation (6.7), since we need to scan the non-zero elements of the difference matrix $\Delta\mathbf{M}^{(t)}$. And the term of $O(\min(\hat{\mathrm{l}}, \hat{\mathrm{n}}) \cdot \mathrm{l}^2)$ comes from the steps 22-23 of *Fast-Batch-Update*. □

6.5 Dynamic Proximity: Applications

In this section, we give the complete algorithms for the two applications we posed in Section 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., Alg. 10) 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, $\mathbf{\Lambda}^{(t)}$; 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, we can easily verify that our algorithms give the exact answers, without any quality loss or approximation.

Algorithm 10 Initialization

Require: The adjacency matrix at time step 1 $\mathbf{M}^{(1)}$, and the parameter c.
Ensure: 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 $\mathbf{\Lambda}^{(1)}$.
1: get the fixed degree matrix $\tilde{\mathbf{D}}$ as equation (6.4)
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: $\mathbf{\Lambda}^{(1)} = (\mathbf{I} - c^2\mathbf{Mc}^{(1)} \cdot \mathbf{Mr}^{(1)})^{-1}$
5: store the matrices: $\mathbf{Mr}^{(1)}$, $\mathbf{Mc}^{(1)}$, and $\mathbf{\Lambda}^{(1)}$.

6.5.1 *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 (equation 6.3) and the fast update algorithms we developed in Section 4, we can get the following algorithm (Alg. 11) to track the top-k queries over time. The algorithm for tracking centrality for a single query node is quite similar to Alg. 11. We omit the details for space.

Algorithm 11 *Track-Centrality* (Top-*k* Queries)

Require: The time-evolving bipartite graphs $\{\mathbf{M}^{(1)}, \Delta\mathbf{M}^{(t)}(t \geq 2)\}$, the parameters c and k
Ensure: 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: $x = \mathbf{1}_{1\times n} \cdot \mathbf{Mr}^{(t)} \cdot \mathbf{\Lambda}^{(\mathbf{t})}$; and $y = \mathbf{1}_{1\times l} \cdot \mathbf{\Lambda}^{(\mathbf{t})}$
4: $\vec{r_2}' = c \cdot x + y$
5: $\vec{r_1}' = c \cdot \vec{r_2'} \cdot \mathbf{Mc}^{(t)}$
6: output the top *k* type 1 objects according to $\vec{r_1}'$ (larger value means more central)
7: output the top *k* type 2 objects according to $\vec{r_2}'$ (larger value means more central)
8: Update $\mathbf{Mr}^{(t)}$, $\mathbf{Mc}^{(t)}$, and $\mathbf{\Lambda}^{(\mathbf{t})}$ for $t \geq 2$.
9: **end for**

In step 8 of Alg. 11, 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 $\mathbf{\Lambda}^{(\mathbf{t})}$. 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 13. Correctness of *Track-Centrality*. *The vectors $\vec{r_1}'$ and $\vec{r_2}'$ in Alg. 11 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 equation* (6.3).

Proof: Based on Delta Matrix Inversion Theorems, we have that step 8 of *Track-Centrality* maintains the correct core matrix at each time step.

Apply the Sherman-Morrison Lemma [PC90] to equation (6.2), we have

$$\mathbf{Q}^{(t)} \propto \begin{pmatrix} \mathbf{I} + c^2\mathbf{Mr}^{(t)}\mathbf{\Lambda}^{(\mathbf{t})}\mathbf{Mc}^{(t)} & c\mathbf{Mr}^{(t)}\mathbf{\Lambda}^{(\mathbf{t})} \\ c\mathbf{\Lambda}^{(\mathbf{t})}\mathbf{Mc}^{(t)} & \mathbf{\Lambda}^{(\mathbf{t})} \end{pmatrix} \tag{6.9}$$

By equation (6.3), 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 $\vec{r} = [\text{centrality}(j)]_{j=1,\ldots,(n+l)}$, we have

$$\begin{aligned}
\vec{r}' &\propto [\mathbf{1}_{1\times n},\ \mathbf{1}_{1\times l}] \cdot \mathbf{Q}^{(t)} \\
&\propto \begin{pmatrix} c^2\mathbf{1}_{1\times n}\mathbf{M}\mathbf{r}^{(t)}\mathbf{\Lambda}^{(\mathbf{t})}\mathbf{M}\mathbf{c}^{(t)} + c\mathbf{1}_{1\times l}\mathbf{\Lambda}^{(\mathbf{t})}\mathbf{M}\mathbf{c}^{(t)} \\ c\mathbf{1}_{1\times n}\mathbf{M}\mathbf{r}^{(t)}\mathbf{\Lambda}^{(\mathbf{t})} + \mathbf{1}_{1\times l}\mathbf{\Lambda}^{(\mathbf{t})} \end{pmatrix}' \\
&= \begin{pmatrix} c^2 x\mathbf{M}\mathbf{c}^{(t)} + cy\mathbf{M}\mathbf{c}^{(t)} \\ cx + y \end{pmatrix}' \\
&= [c\vec{r_2}'\mathbf{M}\mathbf{c}^{(t)},\ \vec{r_2}'] \\
&= [\vec{r_1}',\ \vec{r_2}']
\end{aligned}$$

where x and y are two vectors as defined in step 3 of *Track-Centrality* and $\vec{r_1}$,and $\vec{r_2}$ are two column vectors as defined in steps 4-5 of *Track-Centrality*This completes the proof. □

6.5.2 *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?*"

The algorithm for top-k queries is summarized in Alg. 12. The algorithm for tracking the proximity for a given pair of nodes is quite similar to Alg. 12. We omit its details for space.

In Alg. 12, 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 14. Correctness of *Track-Proximity*. *The vectors $\vec{r_1}'$ and $\vec{r_2}'$ in Alg. 12 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* (6.4).

Proof: Based on Delta Matrix Inversion Theorems, we have that step 13 of *Track-Proximity* maintains the correct core matrix at each time step. Therefore, Alg. 7 in step 8 always gives the correct proximity score, which completes the proof. □

Algorithm 12 *Track-Proximity* (Top-*k* Queries)

Require: 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.

Ensure: 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**
    $r_{s,i} =$ **GetQij**$(\mathbf{\Lambda}^{(t)}, \mathbf{Mr}^{(t)}, \mathbf{Mc}^{(t)}, s, i, c))$
  end for
  let $\vec{r_1} = [r_{s,i}](i = 1, ...n)$
  for $j = 1 : l$ **do**
    $r_{s,j} =$ **GetQij**$(\mathbf{\Lambda}^{(t)}, \mathbf{Mr}^{(t)}, \mathbf{Mc}^{(t)}, s, j + n, c))$
  end for
  let $\vec{r_2} = [r_{s,j}](j = 1, ...l)$
  output the top *k* type 1 objects according to $\vec{r_1}'$ (larger value means more relevant)
  output the top *k* type 2 objects according to $\vec{r_2}'$ (larger value means more relevant)
  update $\mathbf{Mr}^{(t)}$, $\mathbf{Mc}^{(t)}$, and $\mathbf{\Lambda}^{(\mathbf{t})}$ for $t \geq 2$.
end for

6.6 Experimental Results

In this section we present experimental results, after we introduce the datasets in subsection 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)?

6.6.1 Data Sets.

We use five different data sets in our experiments, summarized in Table 6.6.1. 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.

The first data set (*NIPS*) is from the NIPS proceedings[3]. The timestamps are publication years, so each graph slice $\mathbf{M}$ corresponds to one 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.

[3] `http://www.cs.toronto.edu/~roweis/data.html`

Table 6.2: Datasets 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 *DM*, *AC*, and *ACPost* data sets are from DBLP[4]. For the first two, we use paper publication years as timestamps, similar to *NIPS*. Thus each graph slice $\mathbf{S}$ corresponds to one year.

DM uses author-paper information for each year between 1995–2007, 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 1959–2007. In contrast to *DM*, columns represent conferences and edges connect authors to conferences they have published in. Each edge in $\mathbf{S}$ 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 $\mathbf{S}$ corresponds to one day, between 2002-01-03 and 2007-08-24. *ACPost* is otherwise similar to *AC*, with number of papers as edge weights. There 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[5]. Rows represent users and columns represent movies. If a user has rated a particular movie, we connect them with an unweighted edge. This dataset consists of one slice and we use it in subsection 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.

6.6.2 Effectiveness: Case Studies

Here, we show the experimental results for the three applications on real datasets, all of which are consistent with our intuition.

[4] http://www.informatik.uni-trier.de/~ley/db/
[5] http://www.netflixprize.com/

Table 6.3: Top-10 most influential (central) authors up to each year. Note that the top-10 most influential (central) authors change over years.

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'

Results on *Track-Centrality*

We apply Alg. 11 to the *NIPS* dataset. 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 6.3 lists the results for every two 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 Univ. 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. Fig. 6.1(a) plots the centrality ranking for some authors over the years. Again, the 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).

Results on *Track-Proximity*.

We first report the results on the *DM* dataset. 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. 6.4 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-

Table 6.4: Top-5 most related authors for 'Jian Pei' up to each year. Note that the most related authors for 'Jian Pei' change over years.

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'

Buffalo; (3) the IBM-Watson collaboration ('Philip S. Yu' and 'Haixun Wang') got stronger over time.

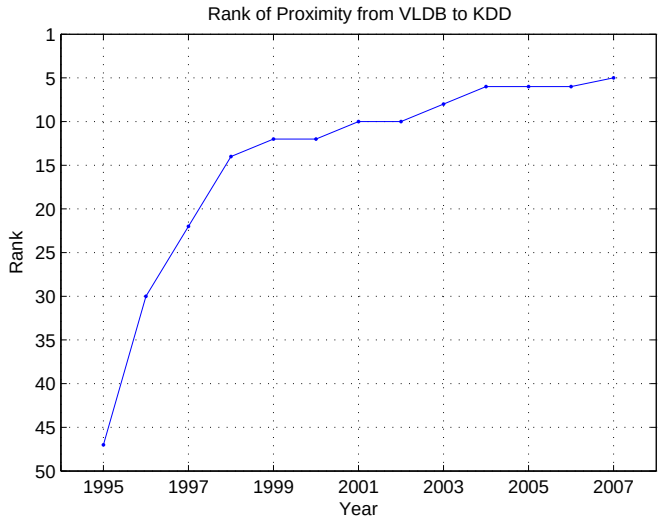


Figure 6.2: The rank of the proximity from 'VLDB' to 'KDD' up to each year

Then, we apply *Track-Proximity* on the dataset *AC*. Here, we want to track the proximity ranking for a given pair of nodes over time. Fig. 6.2 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 $\text{len} = 5$) 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. Fig. 6.1(b)

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 'VLDB') are still one of his major research interests, data mining became a strong research focus ('KDD', 'SDM' and 'ICDM').

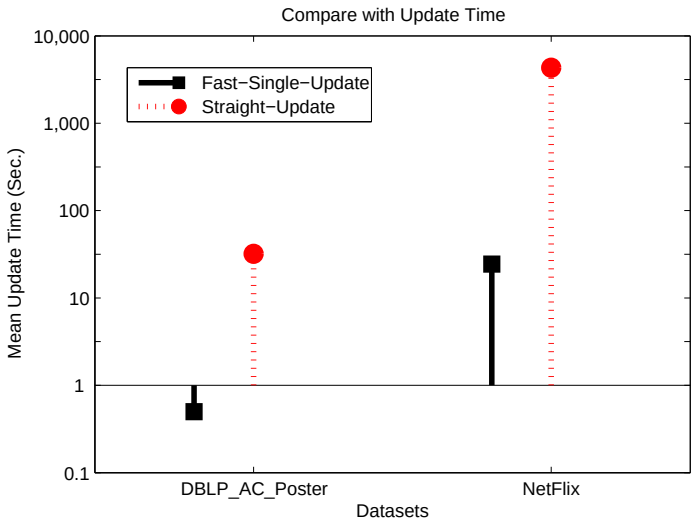


Figure 6.3: Evaluation of *Fast-Single-Update*. For both datasets, one edge changes at each time step. The running time is averaged over multiple runs of experiments and shown in logarithmic scale.

6.6.3 Efficiency

After initialization, at each time step, most time is spent on updating the core matrix $\mathbf{\Lambda}^{(t)}$, as well as the normalized adjacency matrices. In this subsection, we first report the running time for update and then give the total running time for each time step. We use the two largest datasets (*ACPost* and *NetFlix*) to measure performance.

Update Time

We first evaluate *Fast-Single-Update*. Both *ACPost* and *NetFlix* are used. For each dataset, 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. 6.3—Note that the y-axis is in log-scale). On both datasets, *Fast-Single-Update* requires significantly less computation: on *ACPost*, it

is 64x faster (0.5 seconds vs. 32 seconds), while on *NetFlix*, it is 176x faster (22.5 seconds vs $4,313$ seconds).

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 average of 33. The results are summarized in Fig 6.4. We plot the mean update time vs. the number ($\hat{m}$) of changed edges in Fig 6.4(a) and the mean update time vs. the rank ($\min(\hat{n}, \hat{l})$) of the update matrix in Fig 6.4(b). Compared to the Straight-Update, *Fast-Batch-Update* is again much faster, achieving 5–32x speed-up. On average, it is 15x faster than Straight-Update.

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* dataset with one edge changed at each time step and *ACPost* dataset with multiple edges changed at each time step.

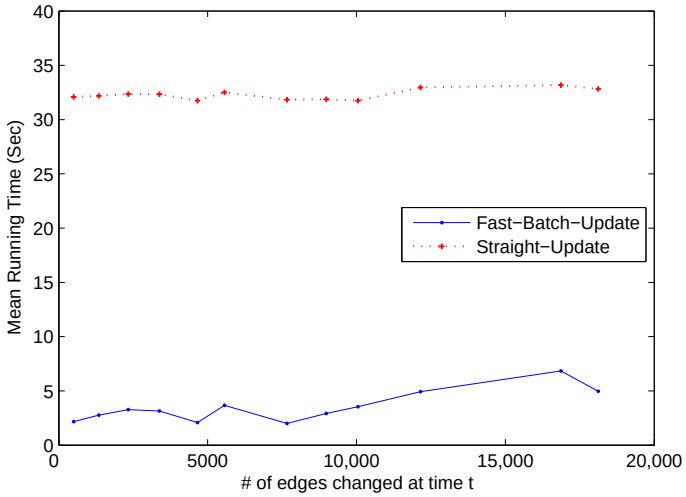
We compare our algorithms ("*Track-Centrality*") with (i) the one that uses Straight-Update in our algorithms (still referred as "Straight-Update"); and (ii) that uses iterative procedure [SQCF05] to compute proximity and centrality at each time step (referred as 'Ite-Alg"). The results are summarized in Fig. 6.5. We can see that in either case, our algorithm (*Track-Centrality*) is much faster. For example, it takes 27.8 seconds on average on the *NetFlix* dataset, which is 155x faster over"Straight-Update" (4,315 seconds); and is 93x faster over "Ite-Alg" (2,582 seconds). In either case, the update time for *Track-Centrality* dominates the overall running time. For example, on the *ACPost* dataset, update time accounts for more than 90% of the overall running time (2.4 seconds vs. 2.6 seconds). 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.

6.7 Related Work

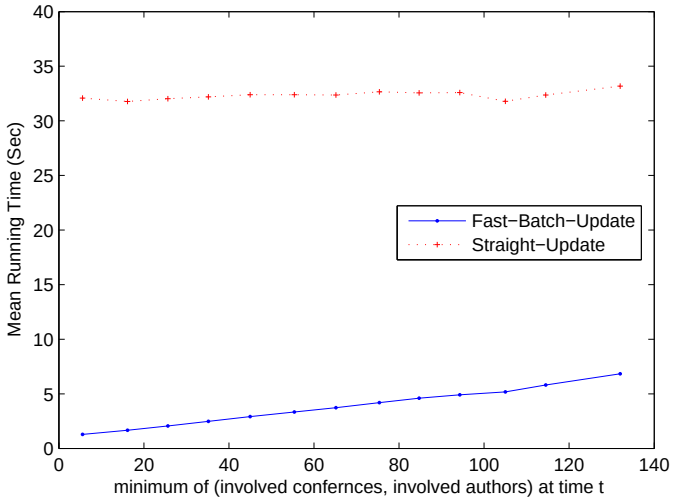
In this section, we review the related work, which can be categorized into two parts: static graph mining and dynamic graph mining.

Static Graph Mining. There is a lot of research work on static graph mining, including pattern and law mining [AJB99, DM02, FFF99, BKM+00, New03], frequent substructure discovery [XHYC05], influence propagation [KKT03], and community mining [FLGC02][GKR98][GN].

In terms of centrality, Google's PageRank algorithm [PBMW98] 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 [HLZ+04, PYFD04, TFP08]. The proposed *Track-Proximity* is its generalization for dynamic bipartite graphs. Other representative



(a) Running Time vs. $\hat{m}$



(b) Running Time vs. $\min(\hat{n}, \hat{l})$

Figure 6.4: Evaluation on *Fast-Batch-Update*.

proximity measurements on static graphs include the sink-augmented delivered current [FMT04], cycle free effective conductance [KNV06], survivable network [GMS93], and direction-aware proximity [TKF07]. Although we focus on random walk with restart in this paper, our fast algorithms can be easily adapted to other random walk based measurements, such as [FMT04, TKF07]. Also, there are a lot of applications of proximity measurements. Representative work includes connection subgraphs [FMT04, KNV06, TF06], neighborhood formation in bipartite graphs [SQCF05], content-based image retrieval [HLZ+04], cross-modal correlation discovery [PYFD04], the BANKS system [ABC+02], link prediction [LNK03], pattern matching [TFGER07], detecting anomalous nodes and links in a graph [SQCF05], ObjectRank [BHP04] and RelationalRank [GMT04].

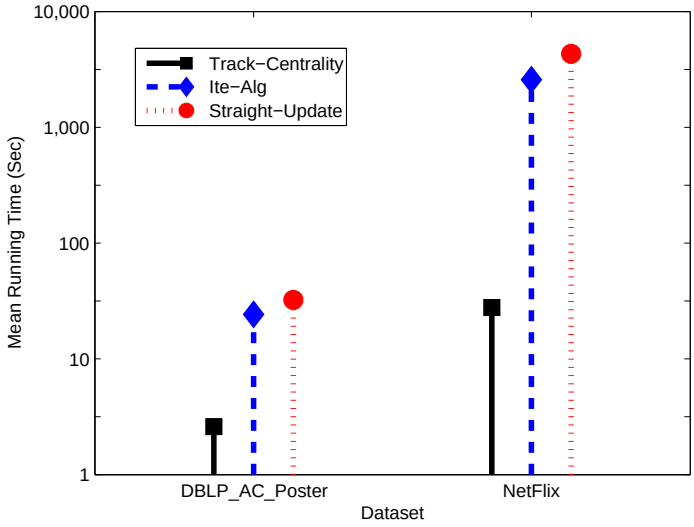


Figure 6.5: Overall running time at each time step for *Track-Centrality*. For *ACPost*, there are multiple edges changed at each time step; and for *NetFlix*, there is only one edge changed at each time step. The running time is averaged in multiple runs of experiments and it is in the logarithm scale

Dynamic Graph Mining. More recently, there is an increasing interest in mining time-evolving graphs, such as densification laws and shrinking diameters [LKF05], community evolution [BHKL06], dynamic tensor analysis [STF06], and dynamic communities [CSZ+07, SFPY07]. 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 [MC06]. However, we have a different setting and focus compared with [MC06]: 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 [MC06] the authors focus on efficiently using time information by adding time as explicit nodes in the graph.

6.8 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. The major contributions of the paper include:

1: Proximity and centrality definitions for time-evolving graphs.

2: 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 4).

3: Two algorithms to incrementally track centrality (*Track-Centrality*) and proximity (*Track-Proximity*), in any-time fashion.

4: Extensive experimental case-studies on several real datasets, showing how different queries can be answered, achieving up to **15∼176x** speed-up.

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.

Part IV

Mining Static Graphs

Chapter 7

Vulnerability Analysis

Summary of This Chapter

- **Questions we want to answer:**

  Q1: How to measure the vulnerability of the graph (by a single number)?

  Q2: Given the vulnerability measurement, how to quantify the backboneness score of a given set of nodes in the graph, i.e., how important are they in terms of maintaining the vulnerability of the graph?

  Q3: Given the backboneness score, how to quickly detect the k nodes that collectively exhibit the highest backboneness score on large, disk-resident graphs?

- **Our answers and contributions**

  A1: We proposed a novel vulnerability measurement for the graph, motivated from immunology and graph loop capacity.

  A2: We proposed a novel definition of backboneness score for a set of nodes, by carefully using the results from the theory of matrix perturbation.

  A3: We proposed a near-optimal and scalable algorithm (NetShield) to find a set of nodes with highest backboneness score, by carefully using results from the theory of sub-modularity.

7.1 Introduction

How to measure the 'Vulnerability' of a given graph? (e.g., *how likely will an epidemic break out given the strength of the virus attack?*) Given a set of k nodes in the graph, how to measure their 'Backboneness' i.e., *how important are they in terms of maintaining the 'Vulnerability' of the whole graph?* How to quickly find k nodes with the highest 'Backboneness' score? This is the core problem behind a lot of important applications. To name a few, the 'Vulnerability' measure

can be used (as one of many other criteria) to evaluate the network design. In the immunization setting, the k with the highest 'Backboneness' scores might be the ones we want quarantine in order to stop an epidemic. Similarly, in the graph demolition setting, these nodes are the ones we want to delete from the graph.

In this chapter, we study this problem in multiple dimensions, by addressing the following three questions:

Q1. How to measure the 'Vulnerability' of the graph (by a single number)?

Q2. Given the 'Vulnerability' measurement, how to quantify the 'Backboneness' score of a given set of nodes in the graph, i.e., how important are they in terms of maintaining the 'Vulnerability' of the graph?

Q3. Given the 'Backboneness' score, how to quickly detect the k nodes that *collectively* exhibit the highest 'Backboneness' score on large, disk-resident graphs?

Here, we focus on exactly these three questions. The main contributions of this chapter are as follows:

1. A novel 'Vulnerability' measurement (λ) for the graph, motivated from immunology and graph loop capacity;
2. A novel definition of 'Backboneness' score $\text{Br}(\mathcal{S})$ for a set of nodes, by carefully using the results from the theory of matrix perturbation;
4. A *near-optimal* and *scalable* algorithm (NetShield) to find a set of nodes with highest 'Backboneness' score, by carefully using results from the theory of sub-modularity.
5. Justifications, proofs and complexity analysis, showing the intuitions, accuracy and efficiency of the proposed methods.
6. Extensive experiments on several real data sets, showing the effectiveness and efficiency of the proposed methods. For the effectiveness, our methods (1) lead to an effective immunization strategy, and (2) always give the mining results which are consistent with human intuitions. For the efficiency, our algorithm (1) achieves significant speed-up over straightforward solutions (*up to 7 orders of magnitude speedup*); and (2) is scalable for large graphs (*linear* wrt the size of the graph).

The rest of the chapter is organized as follows: We give the problem definitions in Section 7.2. We present the proposed 'Vulnerability' measurement and 'Backboneness' score in Section 7.3 and Section 7.4, respectively. We deal with the computational issues in Section 7.5. We evaluate the proposed methods in Section 7.6. We review the related work in Section 7.7 and conclude the chapter in Section 7.8.

7.2 Problem Definitions

Table 7.1 lists the main symbols we use throughout this chapter. In this chapter, we focus on undirected un-weighted graphs. We represent the graph by its adjacency matrix. Following standard notations, we use capital bold letters for matrices (e.g., $\mathbf{A}$), lower-case bold letters for vectors (e.g., $\mathbf{a}$), and calligraphic fonts for sets (e.g., $\mathcal{S}$). We denote the transpose with a prime (i.e., $\mathbf{A}'$ is the

transpose of $\mathbf{A}$), and we use parenthesized superscripts to denote the corresponding variable after deleting the nodes indexed by the superscripts. For example, λ is the first eigen-value of $\mathbf{A}$, then λ^i is the first eigen-value of $\mathbf{A}$ after deleting its $i^{\text{(th)}}$ row/column. We use $(\lambda_i, \mathbf{u}_i)$ to denote the i^{th} eigen-pair (sorted by the magnitude of the eigenvalue) of $\mathbf{A}$. When the subscript is omitted, we refer to them as the first eigenvalue and eigenvector respectively (i.e., $\lambda \triangleq \lambda_1$ and $\mathbf{u} \triangleq \mathbf{u}_1$).

Table 7.1: Symbols

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}, \ldots$	matrices (bold upper case)
$\mathbf{A}(i, j)$	the element at the i^{th} row and j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}(i, :)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	the j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{a}, \mathbf{b}, \ldots$	column vectors
$\mathcal{S}, \mathcal{T}, \ldots$	sets (calligraphic)
n	number of nodes in the graph
m	number of edges in the graph
$(\lambda_i, \mathbf{u}_i)$	the i^{th} eigen-pair of $\mathbf{A}$
λ	first eigen-value of $\mathbf{A}$ (i.e., $\lambda \triangleq \lambda_1$)
$\mathbf{u}$	first eigen-vector of $\mathbf{A}$ (i.e., $\mathbf{u} \triangleq \mathbf{u}_1$)
$\lambda^{(i)}$, $\lambda^{(\mathcal{S})}$	first eigen-value of $\mathbf{A}$ by deleting the node i (or the set of nodes in $\mathcal{S}$)
$\Delta\lambda(i)$	eigen-drop: $\Delta\lambda(i) = \lambda - \lambda^{(i)}$
$\Delta\lambda(\mathcal{S})$	eigen-drop: $\Delta\lambda(\mathcal{S}) = \lambda - \lambda^{(\mathcal{S})}$
Br(i)	'Backboneness' score of node i
Br(S)	'Backboneness' score of nodes in $\mathcal{S}$
V($\mathbf{G}$)	'Vulnerability' score of the graph

With the above notations, our problems can be formally defined as follows:

Problem 7. Measuring 'Vulnerability'

Given: *A large un-directed un-weighted connected graph* $\mathbf{A}$*;*

Find: *A single number V*($\mathbf{G}$)*, reflecting the 'Vulnerability' of the whole graph.*

Problem 8. Measuring 'Backboneness'

Given: *A subset* $\mathcal{S}$ *with* k *nodes in a large un-directed un-weighted connected graph* $\mathbf{A}$*;*

Find: *A single number Br*($\mathcal{S}$)*, reflecting the 'Backboneness' of these* k *nodes, in terms of maintaining the 'Vulnerability' of the whole graph.*

Problem 9. Finding Best-k Backbone Nodes

Given: *A large un-directed un-weighted connected graph* $\mathbf{A}$ *with* n *nodes and an integer* k*;*

Find: *A subset* $\mathcal{S}$ *of* k *nodes with the highest 'Backboneness' score among all* $\binom{n}{k}$ *possible subsets.*

In the next three sections, we present the corresponding solutions respectively.

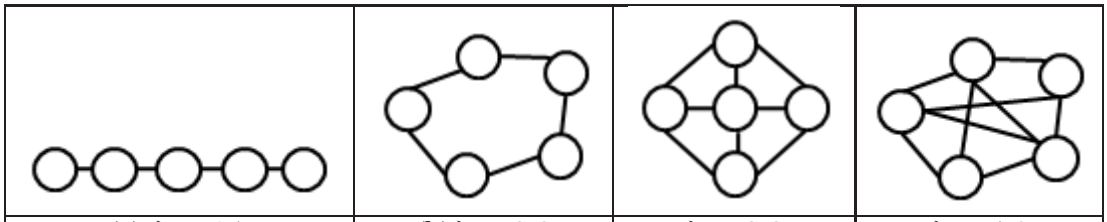
7.3 Our Solution for Problem 7

Here, we focus on Problem 7. We first present our solution and then provide some justifications.

7.3.1 Proposed 'Vulnerability' Score

In Problem 7, the goal is to measure the 'Vulnerability' of the whole graph by a single number. We propose using the first eigenvalue of the adjacency matrix $\mathbf{A}$ as such a measurement (eq. (7.1)): the larger λ is, the more vulnerable the whole graph is.

$$\mathrm{V}(\mathbf{G}) \triangleq \lambda \tag{7.1}$$



(a) $\lambda = 1.7$ (b)$\lambda = 2.0$ $\lambda = 2.9$ $\lambda = 4.0$

Figure 7.1: *An illustrative example of measuring 'Vulnerability' of the graph*

Figure 7.1 presents an illustrative example, where we have four graphs with 5 nodes. Intuitively, from left to right, the vulnerability of the graph increases (i.e., for a given strength of the virus attack, it is more likely that an epidemic will break out in the graphs on the right than those on the left side.). We can see that the corresponding λ increases from left to right as well.

7.3.2 Justifications

Here, we provide some justifications to explain why λ is a good measurement of the graph 'Vulnerability'.

Epidemic Threshold. Our first justification is inspired by immunology. λ is closely related to the epidemic threshold τ of a graph under a flu-like SIR (susceptible-infected-susceptible) epidemic model [CWW+07], and specifically $\tau = 1/\lambda$. This means that a virus less infective than τ will quickly get extinguished instead of lingering forever. Therefore, given the strength of the virus (e.g., the infection rate and the death rate), it is more likely that an epidemic will break out in a graph with larger λ).

Loop Capacity. The second, closely related reason is that λ gives a (approximate) measure of the total number of loops[1] in the graph. Intuitively speaking, the first eigen-value λ contributes most (among all the other eigenvalues) to the number of loops of length l in the graph. To see that, let $\mathrm{LC}(l)$ be the total number of loops with length l in the graph, we have the following equation:

$$\mathrm{LC}(l) = \sum_{i=1}^{n} \lambda_i^l \tag{7.2}$$

[1]A loop in the graph is a path whose starting node is the same as the ending node.

For many real graphs, their spectrum is highly skewed [GMZ03], which means $\lambda \gg \lambda_i (i = 2, ..., n)$. Therefore, $\text{LC}(l)$ is roughly determined by λ (especially when l is big): a larger λ indicators that we have more loops of length l in the graph.

7.4 Our Solution for Problem 8

In this section, we focus on Problem 8. We first present our solution, and then provide justifications. We also discuss and compare our 'Backboneness' measure with some existing node importance measurements in the special case of $k = 1$.

7.4.1 Proposed 'Backboneness' Score

In Problem 8, the goal is to quantify the importance of a given set of nodes in terms of maintaining the 'Vulnerability' of the whole graph. We propose using $\text{Br}(\mathcal{S})$ defined in the following equation.

$$\text{Br}(\mathcal{S}) = \sum_{i \in \mathcal{S}} 2\lambda \mathbf{u}(i)^2 - \sum_{i,j \in \mathcal{S}} \mathbf{A}(i,j)\mathbf{u}(i)\mathbf{u}(j) \tag{7.3}$$

Intuitively, by eq. (7.3), a set of nodes $\mathcal{S}$ has higher 'Backboneness' score if (1) each of them has a high eigen-score ($\mathbf{u}(i)$), and (2) they are dissimilar with each other (small or zero $\mathbf{A}(i,j)$). Figure 7.2 shows some examples on measuring the 'Backboneness' score of a given set of nodes. The best $k = 4$ nodes found by our NetShield (which will be introduced very soon in the next section) are shaded. The result is consistent with intuitions, deleting these nodes will make the graph the least vulnerable (i.e., the remaining graphs are sets of isolated nodes in these examples).

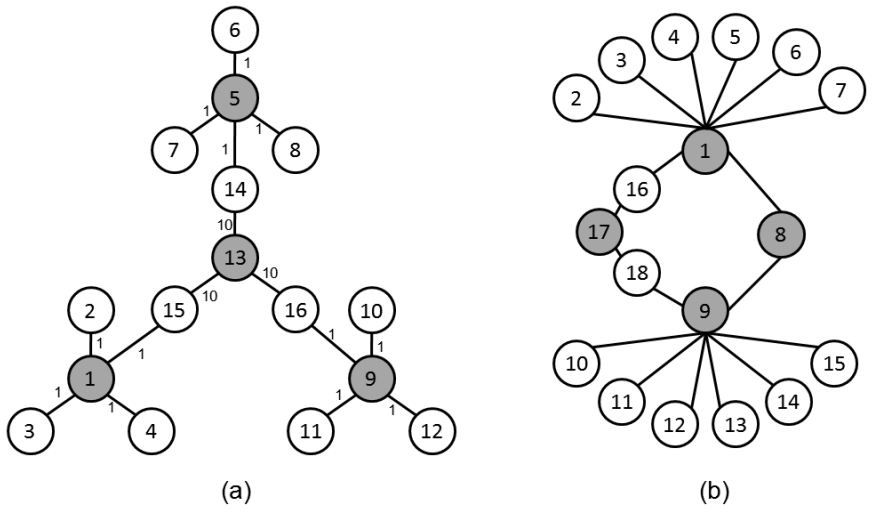


Figure 7.2: *Some examples on measuring the 'Backboneness' score of a given set of nodes. The best $k = 4$ nodes found by our NetShield are shaded.*

7.4.2 Justification

Here, we provide some justifications on the proposed 'Backboneness' score, which is summarized in Lemma 15. It says that $\mathrm{Br}(\mathcal{S})$ is a good approximation for the eigen-drop $\Delta\lambda(\mathcal{S})$ when deleting the set of nodes $\mathcal{S}$ from the original graph $\mathbf{A}$.

Lemma 15. *Let $\lambda^{(\mathcal{S})}$ be the (exact) first eigen-value of $\hat{\mathbf{A}}$, where $\hat{\mathbf{A}}$ is the perturbed version of $\mathbf{A}$ by removing all of its rows/columns indexed by set $\mathcal{S}$. If λ is the simple first eigen-value of $\mathbf{A}$, then $\Delta\lambda(\mathcal{S}) = \lambda - \lambda^{(\mathcal{S})}$ is upper bounded by $Br(\mathcal{S}) + O(\sum_{j\in\mathcal{S}} \|\mathbf{A}(:,j)\|^2)$, where $Br(\mathcal{S})$ is computed by eq. (7.3).*

Proof: First, let us write $\hat{\mathbf{A}}$ as a perturbed version of the original matrix $\mathbf{A}$:

$$\hat{\mathbf{A}} = \mathbf{A} + \mathbf{E}, \quad \text{and} \quad \mathbf{E} = \mathbf{F} + \mathbf{F}' + \mathbf{G} \tag{7.4}$$

where $\mathbf{F}(:,j) = -\mathbf{A}(:,j)$ $(j \in \mathcal{S}$ and $\mathbf{F}(:,j) = 0$ $(j \notin \mathcal{S})$; $\mathbf{G}(i,j) = \mathbf{A}(i,j)$ $(i,j \in \mathcal{S})$ and $\mathbf{G}(i,j) = 0 (i \notin \mathcal{S}, \text{ or } j \notin \mathcal{S})$.

Since $\mathbf{Au} = \lambda\mathbf{u}$, we have

$$\begin{aligned} \mathbf{u}'\mathbf{F}'\mathbf{u} &= \mathbf{u}'\mathbf{F}\mathbf{u} = (\mathbf{F}'\mathbf{u})'\mathbf{u} = -\sum_{j\in\mathcal{S}} \lambda\mathbf{u}(j)^2 \\ \mathbf{u}'\mathbf{G}\mathbf{u} &= \sum_{i,j\in\mathcal{S}} \mathbf{A}(i,j)\mathbf{u}(i)\mathbf{u}(j) \end{aligned} \tag{7.5}$$

Let $\tilde{\lambda}$ be the corresponding perturbed eigen-value of λ, according to the matrix perturbation theory [SS90], we have

$$\begin{aligned} \tilde{\lambda} &= \lambda + \mathbf{u}'\mathbf{E}\mathbf{u} + O(\|\mathbf{E}\|^2) \\ &= \lambda + \mathbf{u}'\mathbf{F}\mathbf{u} + \mathbf{u}'\mathbf{F}'\mathbf{u} + \mathbf{u}'\mathbf{G}\mathbf{u} + O(\|\mathbf{E}\|^2) \\ &= \lambda - (\sum_{j\in\mathcal{S}} 2\lambda\mathbf{u}(j) - \sum_{i,j\in\mathcal{S}} \mathbf{A}(i,j)\mathbf{u}(i)\mathbf{u}(j)) \\ &\quad + O(\sum_{j\in\mathcal{S}} \|\mathbf{A}(:,j)\|^2) \\ &= \lambda - \mathrm{Br}(\mathcal{S}) + O(\sum_{j\in\mathcal{S}} \|\mathbf{A}(:,j)\|^2) \end{aligned} \tag{7.6}$$

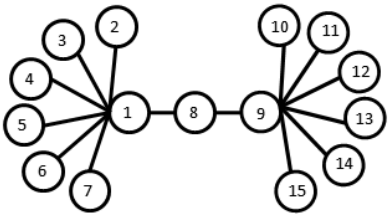
Since $\lambda^{(\mathcal{S})}$ is the first eigen-value of $\hat{\mathbf{A}}$, we have $\lambda^{(\mathcal{S})} \geq \tilde{\lambda}$. Therefore,

$$\begin{aligned} \Delta\lambda(\mathcal{S}) &= \lambda - \lambda^{(\mathcal{S})} \leq \lambda - \tilde{\lambda} \\ &= \mathrm{Br}(\mathcal{S}) + O(\sum_{j\in\mathcal{S}} \|\mathbf{A}(:,j)\|^2) \end{aligned} \tag{7.7}$$

which completes the proof. □

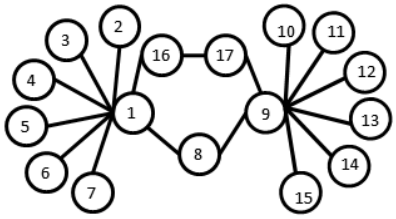
7.4.3 Comparisons in the Case of $k = 1$

In literature, there are a lot of node importance scores (such as PageRank, HITS, betweenness centrality, etc). Our 'Backboneness' score is *fundamentally* different from these node importance scores, in the sense that they *all* aim to measure the importance of an individual node; whereas our 'Backboneness' tries to *collectively* measure the importance of a set of nodes.



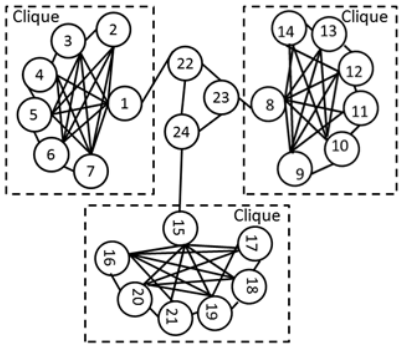
Node	Rank					
	Deg	Short	N. RW	PR	$\Delta\lambda$	ABN
8	2	2	2	2	1	2
1	1	1	1	1	2	1
2	3	3	3	3	3	3

Figure 7.3: *Examples of the 'Backboneness' score of an individual node. (a)*



Node	Rank					
	Deg	Short	N. RW	PR	$\Delta\lambda$	ABN
8	2	2	2	3	1	3
1	1	1	1	1	2	1
16	2	3	3	2	3	2
2	4	4	4	4	4	4

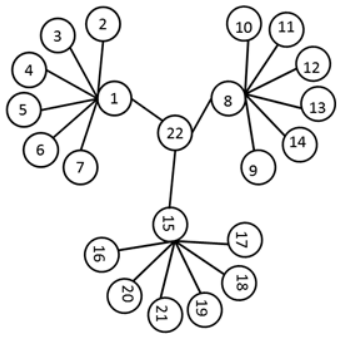
Figure 7.4: *Examples of the 'Backboneness' score of an individual node. (b)*



Node	Rank					
	Deg	Short	N. RW	PR	$\Delta\lambda$	ABN
22	3	1	1	3	1	1
1	1	2	2	1	2	2
2	2	3	3	2	3	3

Figure 7.5: *Examples of the 'Backboneness' score of an individual node. (c)*

Nonetheless, it is interesting to compare them in the special case of $k = 1$. Figures 7.3-7.6show some examples on measuring the 'Backboneness' score of an individual node. We compare it with some possible choices: Degree ('Deg'), Betweeness Centrality based on the shortest path ('Short') [Fre77], Betweeness Centrality based on random walk ('N.RW') [New05], PageRank



Node	Rank					
	Deg	Short	N. RW	PR	$\Delta\lambda$	ABN
22	2	1	1	2	1	2
1	1	2	2	1	2	1

Figure 7.6: *Examples of the 'Backboneness' score of an individual node. (d)*

('PR') [PBMW98], and abnormality score ('ABN') [SQCF05]. We can see that the proposed $\Delta\lambda$ is the only one that is always consistent with intuitions in all the settings. For each figure, the table on the right shows some node indices (the first column) sorted according to human intuitions (the most important node comes first); each of the rest columns shows the rank by the corresponding measurement. Shaded columns are the ones that agree with intuitions (= the first column). Notice that the proposed 'Backboneness' score ($\Delta\lambda$) is the **only one** that consistently agrees with intuitions. Take figure 7.3 as an example, intuitively, node 8 should receive a higher 'Backboneness' score than node 1 since node 8 connects the two communities with each other, whereas node 1 is a local center for the left community. It can be seen that each shortest path which goes through node 8 between two nodes on this graph must also go through node 1. On the other hand, some shortest paths (e.g., the shortest path between node 7 and node 2) only pass node 1 but not node 8. Therefore, by 'Short', node 1 will receive a higher score than node 8 which is counter-intuitive. For the similar reason, 'N.RW' will also think node 1 is more important than node 8. As for 'PR', its score is more or less proportional to the degree in un-directed graphs. Therefore, 'PR' also ranks node 1 higher than node 8. Lastly, as for 'ABN', it measures the abnormality of a given node by looking at its neighborhood: a node is abnormal if its neighborhood is dissimilar with each other. However, in this example, the neighborhood of node 8 (node 1 and node 9) is totally symmetric; whereas that of node 1 (nodes 2-7 and node 8) is not. Hence, 'ABN' again ranks node 1 higher than node 8.

7.5 Our Solution for Problem 9

In this section, we deal with Problem 9. Here, the goal is to find a subset of k nodes with the highest 'Backboneness' score (among all $\binom{n}{k}$ possible subsets). We start by showing that the two straightforward methods (referred to as 'Com-Eigs', and 'Com-Eval') are computationally intractable. Then, we present the proposed NetShield algorithm. Finally, we analyze its accuracy as well as its computational complexity.

7.5.1 Preliminaries

There are two obviously straight-forward methods for Problem 9. The first one (referred to as 'Com-Eigs') works as follows: for each possible subset $\mathcal{S}$, we delete the corresponding rows/columns from the adjacency matrix $\mathbf{A}$; compute the first eigenvalue of the new perturbed adjacency matrix; and finally output the subset of nodes which has the smallest eigenvalue (therefore has the largest eigen-drop). Despite the simplicity of this strategy, it is computational intractable due to its combinatorial nature. It is easy to show that the computational complexity of 'Com-Eigs' is $O(\binom{n}{k} \cdot m)$[2]. This is computationally intractable even for small graphs. For example, in a graph with 1K nodes and 10K edges, suppose that it takes about 0.01 second to find its first eigen-value. Then we need about 2,615 years to find the best-5 nodes with the highest 'Backboneness' score!

A more reasonable (in terms of speed) way to find the best-k nodes is to evaluate $\text{Br}(\mathcal{S})$, rather than to compute the first eigen-value $\lambda_{\mathcal{S}}$, $\binom{n}{k}$ times, and pick the subset with the highest $\text{Br}(\mathcal{S})$. We refer to this strategy as 'Com-Eval'. Compared with the straight-forward method (referred to as 'Com-Eigs', which is $O(\binom{n}{k} \cdot m)$); 'Com-Eval' is much faster ($O(\binom{n}{k} \cdot k^2)$). However, 'Com-Eval' is still not applicable to real applications due to its combinatorial nature. Again, in a graph with 1K nodes and 10K edges, suppose that it only takes about 0.00001 second to evaluate $\text{Br}(\mathcal{S})$ once. Then we still need about 3 months to find the best-5 nodes with the highest 'Backboneness' score!

7.5.2 Proposed "NetShield" Algorithm

The proposed NetShield is given in Alg. 13. In Alg. 13, we compute the first eigenvalue λ and the corresponding eigenvector $\mathbf{u}$ in step 1. In step 4, the $n \times 1$ vector $\mathbf{v}$ measures the 'Backboneness' score of each individual node. Then, in each iteration of steps 6-17, we greedily select one more node and add it into set $\mathcal{S}$ according to score(j) (step 13). Note that steps 10-12 are to exclude those nodes that are already in the selected set $\mathcal{S}$.

7.5.3 Analysis of NetShield

Here, we analyze the accuracy and efficiency of the proposed NetShield.

First, according to the following theorem, Alg. 13 is *near-optimal* wrt 'Com-Eval':

Theorem 5. Effectiveness of NetShield. *Let $\mathcal{S}$ and $\tilde{\mathcal{S}}$ be the sets selected by Alg. 13 and by 'Com-Eval', respectively. Let $\Delta\lambda(\mathcal{S})$ and $\Delta\lambda(\tilde{\mathcal{S}})$ be the corresponding eigen-drops. Then, $\Delta\lambda(\mathcal{S}) \geq (1 - 1/e)\Delta\lambda(\tilde{\mathcal{S}})$.*

Proof: Let $\mathcal{I}, \mathcal{J}, \mathcal{K}$ be three sets and $\mathcal{I} \subseteq \mathcal{J}$. Define the following three sets based on $\mathcal{I}, \mathcal{J}, \mathcal{K}$: $\mathcal{S} = \mathcal{I} \cup \mathcal{K}$, $\mathcal{T} = \mathcal{J} \cup \mathcal{K}$, $\mathcal{R} = \mathcal{I} \setminus \mathcal{J}$.

[2]We assume that k is relatively small compared with n and m (e.g., tens or hundreds). Therefore, after deleting k rows/columns from $\mathbf{A}$, we still have $O(m)$ edges.

Algorithm 13 NetShield

Require: the adjacency matrix $\mathbf{A}$ and an integer k
Ensure: a set $\mathcal{S}$ with k nodes

1: compute the first eigen-value λ of $\mathbf{A}$; let $\mathbf{u}$ be the corresponding eigen-vector $\mathbf{u}(j)(j = 1, ..., n)$;
2: initialize $\mathcal{S}$ to be empty;
3: **for** $j = 1$ to n **do**
4: $\quad \mathbf{v}(j) = (2 \cdot \lambda^2 - \mathbf{A}(j, j)) \cdot u(j)^2$;
5: **end for**
6: **for** iter $= 1$ to k **do**
7: $\quad$ let $\mathbf{B} = \mathbf{A}(:, \mathcal{S})$;
8: $\quad$ let $\mathbf{b} = \mathbf{B} \cdot \mathbf{u}(\mathcal{S})$;
9: $\quad$ **for** $j = 1$ to n **do**
10: $\quad\quad$ **if** $j \in \mathcal{S}$ **then**
11: $\quad\quad\quad$ let score$(j) = -1$;
12: $\quad\quad$ **else**
13: $\quad\quad\quad$ let score$(j) = \mathbf{v}(j) - 2 \cdot \mathbf{b}(j) \cdot \mathbf{u}(j)$;
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad$ let $i =$ argmax$_j$score(j), add i to set $\mathcal{S}$;
17: **end for**
18: return $\mathcal{S}$.

Substituting eq.(7.3), we have

$$\begin{aligned} (\text{Br}(\mathcal{S}) &- \text{Br}(\mathcal{I})) - (\text{Br}(\mathcal{T}) - \text{Br}(\mathcal{J})) \\ &= 2 \sum_{i \in \mathcal{K}, j \in \mathcal{R}} \mathbf{A}(i, j)\mathbf{u}(i)\mathbf{u}(j) \geq 0 \\ &\Rightarrow \text{Br}(\mathcal{S}) - \text{Br}(\mathcal{I}) \geq \text{Br}(\mathcal{T}) - \text{Br}(\mathcal{J}) \end{aligned} \tag{7.8}$$

Therefore, the function $\mathbf{Br}(\mathcal{S})$ is sub-modular.

Next, we can verify that node i selected in step 16 of Alg. 13 satisfies $i = \text{argmax}_{j \notin \mathcal{S}} \text{Br}(\mathcal{S} \cup j)$ for a fixed set $\mathcal{S}$.

Finally, it is clear that $\text{Br}(\phi) = 0$, where ϕ is an empty set. Using the property of sub-modular functions [KG07], we have $\Delta\lambda(\mathcal{S}) \geq (1 - 1/e)\Delta\lambda(\tilde{\mathcal{S}})$. □

According to Lemma 16, the computational complexity of Alg. 13 is $O(nk^2 + m)$, which is much faster than both 'Com-Eigs' ($O(\binom{n}{k}m)$) and 'Com-Eval' ($O(\binom{n}{k}k^2)$).

Lemma 16. Efficiency of NetShield. *The computational complexity of Alg. 13 is* $O(nk^2 + m)$.

Proof: The cost of step 1 is $O(m)$, and the cost of step 2 is constant. For steps 3-5, its cost is $O(n)$.

For each inner loop of steps 6-17, its cost is $O(n) + O(n \cdot \text{iter})$. Therefore, we have

$$\begin{aligned} \text{cost}(NetShield) &= O(m) + O(n) + \sum_{\text{iter}=1}^{k} (n + n \cdot \text{iter}) \\ &= O(nk^2 + m) \end{aligned} \tag{7.9}$$

which completes the proof. □

7.6 Experimental Evaluations

We present detailed experimental results in this section. All the experiments are designed to answer the following questions:

1: (*Effectiveness*) How effective is the proposed $\text{Br}(\mathcal{S})$ in real graphs?

2: (*Efficiency*) How fast and scalable is the proposed NetShield?

7.6.1 Data sets

Table 7.2: Summary of the data sets

Name	n	m
Karate	34	152
AA	418,236	2,753,798
NetFlix	2,667,199	171,460,874

We used three real data sets, which are summarized in table 7.2. The first data set (*Karate*) is a unipartite graph, which describes the friendship among the 34 members of a karate club at a US university [Zac77]. Each node is a member in the karate club and the existence of the edge indicates that the two corresponding members are friends. Overall, we have $n = 34$ nodes and $m = 156$ edges.

The second data set (*AA*) is from DBLP.[3] *AA* is a co-authorship network, where each node is an author and the existence of an edge indicates the co-authorship between the two corresponding persons. Overall, we have $n = 418,236$ nodes and $m = 2,753,798$ edges. We also construct much smaller co-authorship networks, using the authors from only one conference (e.g., *NIPS, SIGIR, SIGMOD*, etc.). For example, *NIPS* is the co-authorship network for the authors in the 'NIPS' conference. For these smaller co-authorship networks, they typically have a few thousand nodes and up to a few ten thousand edges.

The last data set (*NetFlix*) is from the Netflix prize.[4] This is also a bipartite graph. We have two types of nodes: user and movie. The existence of an edge indicates that the corresponding user has

[3] `http://www.informatik.uni-trier.de/~ley/db/`
[4] `http://www.netflixprize.com/`

Table 7.3: Evaluation on the approximation accuracy of f($\mathcal{S}$). Larger is better.

k	'KDD'	'ICDM'	'SDM'	'SIGMOD'
1	0.9519	0.9908	0.9995	1.0000
2	0.9629	0.9910	0.9984	0.9927
5	0.9721	0.9888	0.9992	0.9895
10	0.9726	0.9863	0.9987	0.9852
20	0.9683	0.9798	0.9929	0.9772

rated the corresponding movie. Overall, we have $n = 2,667,199$ nodes and $m = 171,460,874$ edges. This is a bipartite graph, and we convert it to a unipartite graph $\mathbf{A}$: $\mathbf{A} = \begin{pmatrix} \mathbf{0} & \mathbf{B} \\ \mathbf{B}' & \mathbf{0} \end{pmatrix}$, where $\mathbf{0}$ is a matrix with all zero entries.

7.6.2 Effectiveness

Approximation quality of Br($\mathcal{S}$)

The proposed NetShield is based on eq. (7.3). That is, we want to approximate the first eigen-value of the perturbed matrix by λ and $\mathbf{u}$. So first, let us evaluate how good this approximation is. We construct an authorship network from one of the following conferences: *'KDD', 'ICDM', 'SDM', 'SIGMOD'*. We then compute the linear correlation coefficient between $\Delta\lambda(\mathcal{S})$ and Br($\mathcal{S}$) with several different k values ($k = 1, 2, 5, 10, 20$). The results are shown in table 7.3. It can be seen that the approximation is very good - in all the cases, the linear correlation coefficient is greater than 0.95.

Accuracy of NetShield

Here, we evaluate the accuracy of the proposed NetShield. For the *Karate* graph, we use the proposed NetShield to find a set of k nodes and check the corresponding eigen-drop (i.e., the decrease of the first eigen-value of the adjacency matrix). We compare it with 'Com-Eigs', which always gives the optimal solutions (i.e., it returns the subset that leads to the largest eigen-drop). We also plot $(1 - \frac{1}{e})$ of the eigen-drop given by 'Com-Eigs' (green dashed curve). The result is plotted in figure 7.7. It can be seen that the proposed NetShield is *near-optimal*: it is always above the green line ($(1-\frac{1}{e})$ of the optimal solution) and often it is very close to the blue line (the optimal solution).

Immunization by NetShield

The proposed 'Vulnerability' score of the graph is partially motivated from the epidemic threshold [CWW+07]. As a consequence, the proposed NetShield leads to a natural immunization strat-

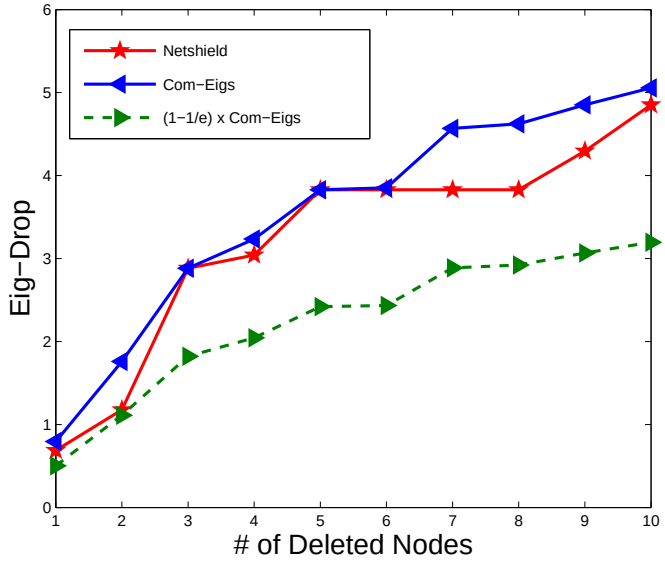


Figure 7.7: *Evaluation of the accuracy of NetShield. Eigen-drop vs. the number of deleted nodes. Higher is better. The proposed NetShield (red star) is near-optimal. Best viewed in color.*

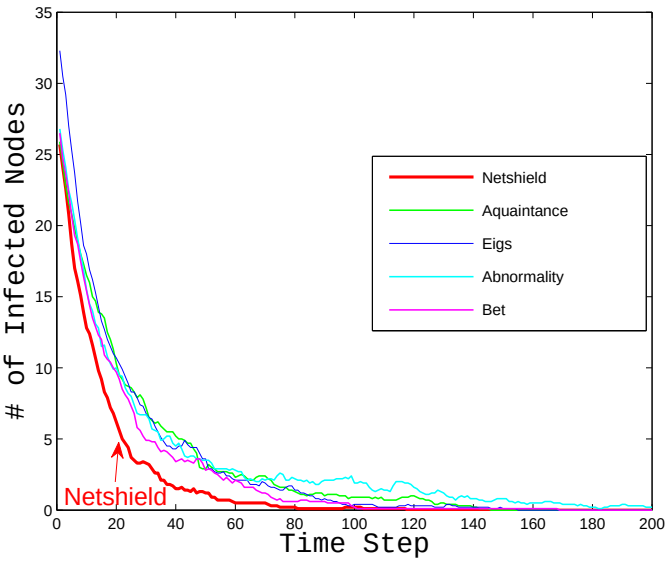


Figure 7.8: *Evaluation of immunization of NetShield on the* Karate *graph ($b = 0.01$ and $d = 0.07$). The number of infected nodes vs. the time step. b and d are the infection rate and the recovery rate, respectively. Lower is better. The proposed NetShield is always the best. Best viewed in color.*

egy for SIS (suspectable-infection-suspectable) model:[5] to quarantine or delete the subset of the

[5]According to [CWW+07], for SIR (suspectable-infection-recovered) model, its epidemic threshold is also determined by λ. Therefore, we expect that our NetShield can also immunize for SIR model.

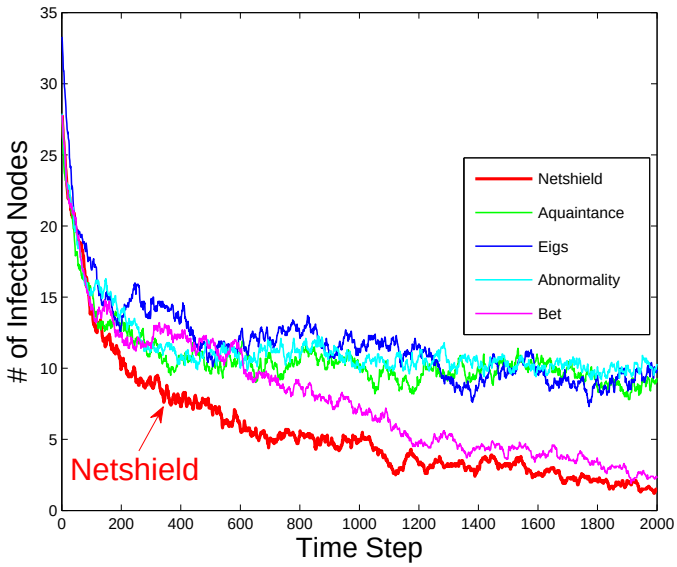


Figure 7.9: *Evaluation of immunization of NetShield on the* Karate *graph (*$b = 0.01$ *and* $d = 0.025$*). The number of infected nodes vs. the time step.* b *and* d *are the infection rate and the recovery rate, respectively. Lower is better. The proposed NetShield is always the best. Best viewed in color.*

nodes detected by NetShield in order to prevent an epidemic from breaking out. We compare it with the following alternative choices: (1) picking a random neighbor of a randomly chosen node[CHbA91] ('Aquaintance'), (2) picking the nodes with the highest eigen scores $\mathbf{u}(i)(i = 1, ..., n)$ ('Eigs'), (3) picking the nodes with the highest abnormality scores [SQCF05] ('abnormality'), and (4) picking the nodes with the highest betweenness centrality scores [Fre77]('Bet'). For each method, we delete 5 nodes for immunization. The result is presented in figures 7.7-7.9, which are averaged over 10 runs. It can be seen that the proposed NetShield is always the best, - its curve is always the lowest which means that we always have the least number of infected nodes in the graph with this immunization strategy. Note that the black curve ('Original') is the one without any immunization strategy.

Case studies

Next, we will show some case studies to illustrate the effectiveness of the proposed $\text{Br}(\mathcal{S})$ as a 'Backboneness' score of a subset of nodes.

Karate. We start with the *Karate* network, which is widely used in social network analysis. In figure 7.11, there are two different communities in the graph (shaded). We first want to measure the 'Backboneness' score for an individual node/member. The first ten nodes with the highest individual 'Backboneness' scores are labeled by their ranks (for example, node 1 has the highest 'Backboneness' score, etc). The result is consistent with intuitions, - they are either the bridges of the two communities (nodes 1, 3, 7 and 9), or the centers of the local communities (nodes 5 and

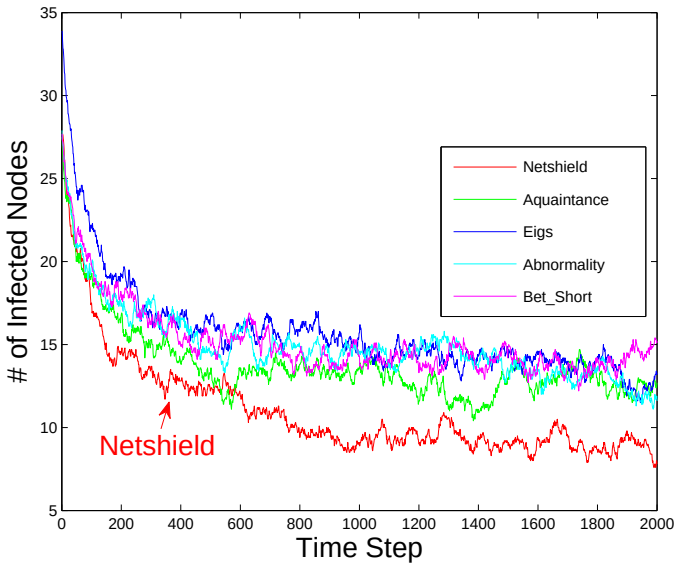


Figure 7.10: *Evaluation of immunization of NetShield on the* Karate *graph (* $b = 0.01$ *and* $d = 0.015$*). The number of infected nodes vs. the time step.* b *and* d *are the infection rate and the recovery rate, respectively. Lower is better. The proposed NetShield is always the best. Best viewed in color.*

10 for the left community; nodes 2, 4, 6, and 7 for the right community). Notice that node 1 has a higher score than node 2 although node 1 has a lower degree than node 1. This is because node 1 is the bridge between the two communities; whereas node 2 is the center only for the nodes in the right community.

Then, we want to measure the 'Backboneness' score for a given set of nodes/members. In figure 7.11, the best $k = 5$ nodes found by NetShield are shown in black. Again, the result is consistent with intuitions. It is interesting to notice that the best subset with 5 nodes (nodes 1, 2, 3, 5 and 10) is different from the first 5 nodes with the highest individual 'Backboneness' scores (nodes 1-5).

AA. We run the proposed NetShield on *AA* data set and return the best $k = 200$ authors. Some representative authors, to name a few, are *'Sudhakar M. Reddy' 'Wei Wang' 'Heinrich Niemann', 'Srimat T. Chakradhar', 'Philip S. Yu', 'Lei Zhang', 'Wei Li', 'Jiawei Han', 'Srinivasan Parthasarathy', 'Srivaths Ravi', 'Antonis M. Paschalis', 'Mohammed Javeed Zaki', 'Lei Li', 'Dimitris Gizopoulos', 'Alberto L. Sangiovanni-Vincentelli', 'Narayanan Vijaykrishnan', 'Jason Cong', 'Thomas S. Huang', etc.* We can make some very interesting observations from the result: (1) There are some multi-disciplinary people in the result. For example, Prof. Alberto L. Sangiovanni-Vincentelli from UC Berkeley is interested in 'design technology', 'cad', 'embedded systems', and 'formal verification'; Prof. Philip S. Yu from UIC is interested in 'databases', 'performance', 'distributed systems' and 'data mining'. (2) Some people show up because they are famous in one specific area, and occasionally have one/two papers in a remotely related area

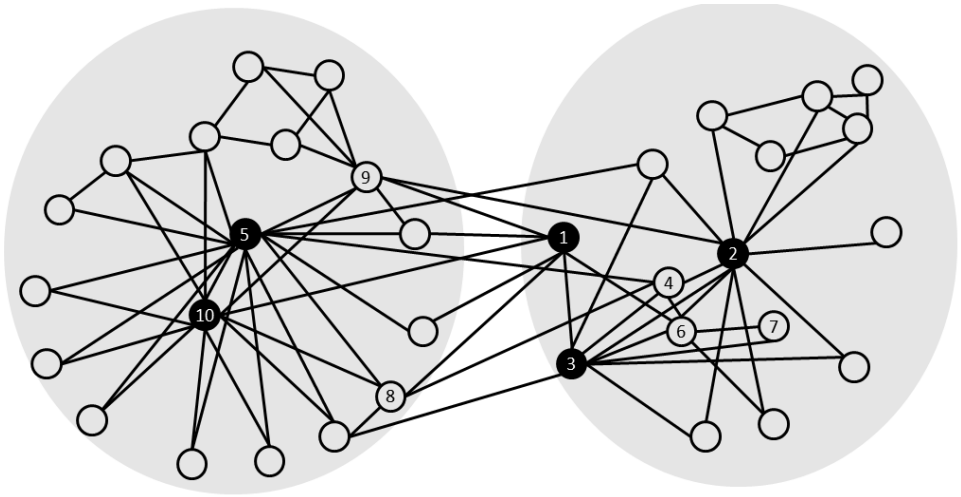


Figure 7.11: Karate *data set: There are two communities (shaded). The top ten nodes with the highest individual 'Backboneness' scores are labeled by their ranks; the best $k = 5$ nodes discovered by NetShield are in black. Notice the agreement of the chosen black nodes with intuitions: removing them would severely disconnect the karate club*

(therefore, bridging two remote areas). For example, Dr. Srimat T. Chakradhar mainly focuses on 'cad'. But he has co-authored in a 'NIPS' paper. Therefore, he is critical to bridge these two (originally) remote areas: 'cad' and 'machine learning'. (3) Some people show up because they have ambiguous names (e.g., Wei Wang, Lei Li, Lei Zhang, Wei Li, etc.). Take 'Wei Wang' as an example; according to DBLP,[6] there are 7 different 'Wei Wang's. In our experiment, we treat all of them as one person. That is to say, it is equivalent to putting an artificial 'Wei Wang' in the graph who is bridging 7 different 'Wei Wang's together. These 7 'Wei Wang's are in fact spread out in quite different areas. (e.g., Wei Wang@UNC is in 'data mining' and 'bio'; Wei Wang@NUS is in 'communication'; Wei Wang@MIT is in 'non-linear systems'.)

NetFlix. We also performed a case study on the *NetFlix* data set. Table 7.4 shows the best $k = 10$ movies found by our NetShield algorithm. The resulting movies are all popular movies which are favored by different types of populations.

7.6.3 Efficiency

We will study the wall-clock running time of the proposed NetShield here. Basically, we want to answer the following two questions:

1. *(Speed)* What is the speedup of the proposed NetShield over the straight-forward methods ('Com-Eigs' and 'Com-Eval')?
2. *(Scalability)* How does NetShield scale with the size of the graph (n and m) and k?

[6] `http://www.informatik.uni-trier.de/~ey/db/indices/a-tree/w/Wang:Wei.html`

Table 7.4: The best $k = 10$ movies from *NetFlix* data set (The 3^{th} columns is the number of Oscar awards/nominations that the corresponding movie won.)

Movie Title	**Genre**	**Oscar**
Pirates of the Caribbean: The Curse of the Black Pearl	Action;Adventure Comedy;Fantasy	5
Forrest Gump	Comedy;Drama;Romance	6
Lord of the Rings: The Fellowship of the Ring	Action;Adventure;Fantasy	4
Lord of the Rings: The Two Towers	Action;Adventure;Fantasy	2
Big Doll House	Drama;Mystery;Thriller	6
The Shawshank Redemption	Drama	7
The Green Mile	Crime;Drama;Fantasy;Mystery	4
Independence Day	Action;Thriller	0
Gladiator	Action;Adventure;Drama	0
The Matrix	Action;Thriller	4

For the results we report in this subsection, all of the experiments are done on the same machine with four 2.4GHz AMD CPUs and 48GB memory, running Linux (2.6 kernel). If the program takes more than 1,000,000 seconds (more than 10 days), we stop running it.

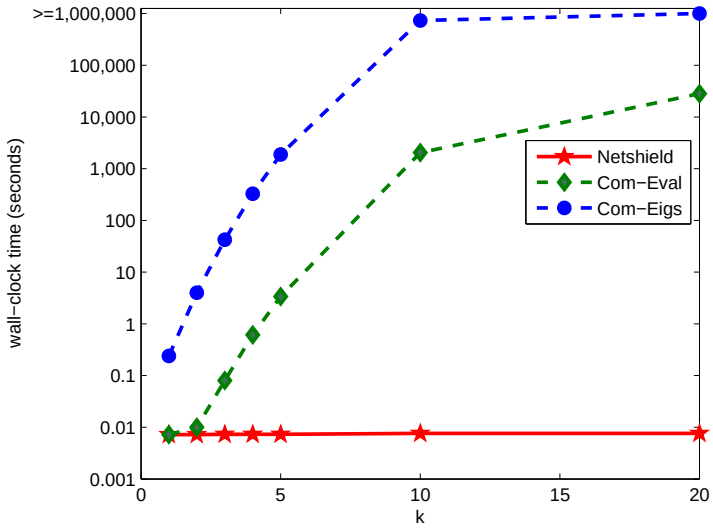


Figure 7.12: *Comparison of speed for different methods on* Karate. *We fix n and m and vary k from 1 to 20. The time is in the logarithm scale. Our NetShield (red star) is much faster. Lower is better.*

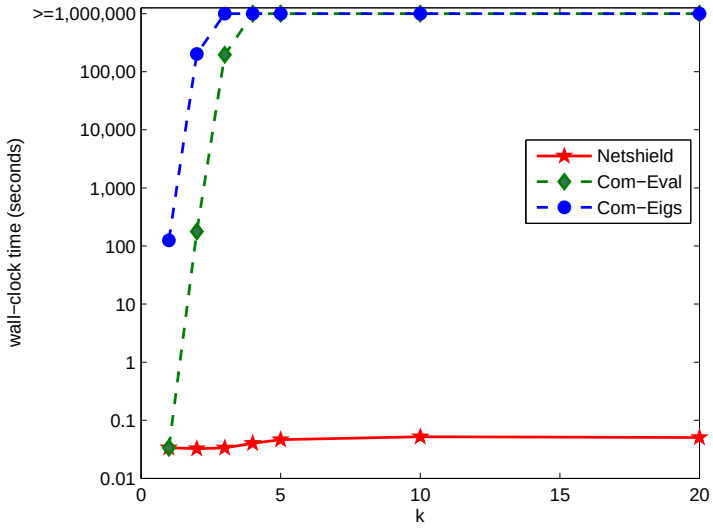


Figure 7.13: *Comparison of speed for different methods on* NIPS. *We fix n and m and vary k from 1 to 20. The time is in the logarithm scale. Our NetShield (red star) is much faster. Lower is better.* NIPS *is the co-authorship network from the NIPS conference.*

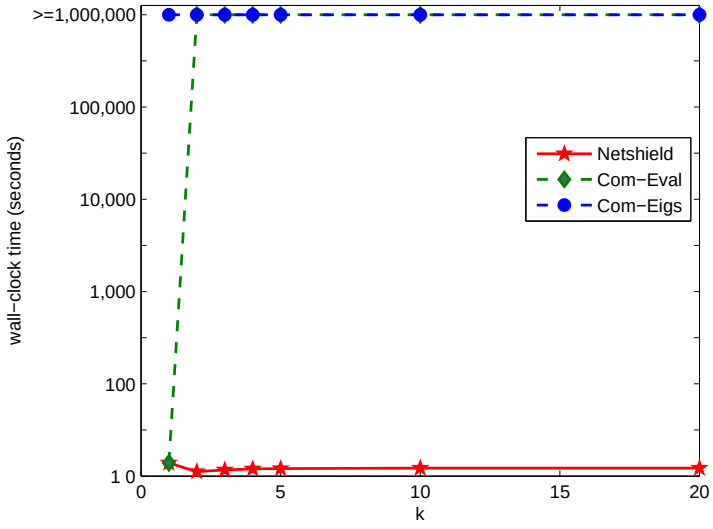


Figure 7.14: *Comparison of speed for different methods on* AA. *For each sub-figure, we fix n and m and vary k from 1 to 20. The time is in the logarithm scale. Our NetShield (red star) is much faster. Lower is better.*

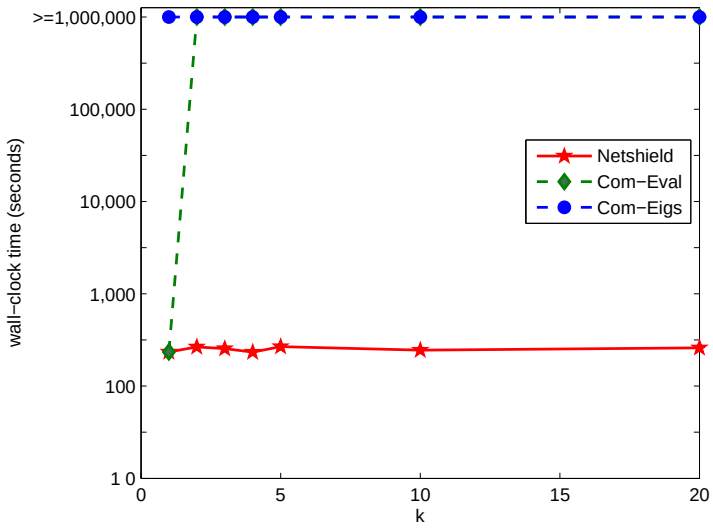


Figure 7.15: *Comparison of speed for different methods on* NetFlix. *For each sub-figure, we fix n and m and vary k from 1 to 20. The time is in the logarithm scale. Our NetShield (red star) is much faster. Lower is better.*

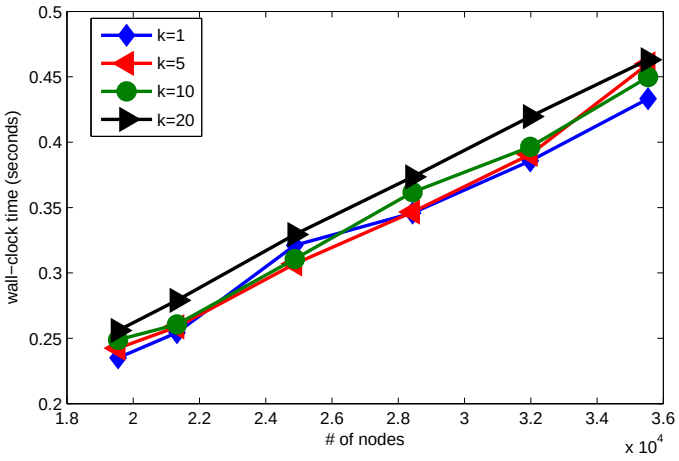
First, we compare NetShield with 'Com-Eigs' and 'Com-Eval'[7]. Figures 7.12, 7.13, 7.14, and 7.15 show the comparison on four real data sets. We can make the following conclusions: (1) Straight-forward methods ('Com-Eigs' and 'Com-Eval') are computationally intractable even for a small graph. For example, on the *Karate* data set with only 34 nodes, it takes more than 100,000 and 1,000 seconds to find the best-10 by 'Com-Eigs' and by 'Com-Eval', respectively. (2) The speedup of the proposed NetShield over both 'Com-Eigs' and 'Com-Eval' is huge - in most cases, we achieve *several (up to 7) orders of magnitude* speedups! (3) The speedup of the proposed NetShield over both 'Com-Eigs' and 'Com-Eval' quickly increases wrt the size of the graph as well as k. (4) For a given size of the graph (fixed n and m), the wall-clock time is almost constant - suggesting that NetShield spends most of its running time in computing λ and $\mathbf{u}$.

Next, we evaluate the scalability of NetShield. From figure 7.16, it can be seen that NetShield scales linearly wrt both n and m, which means that it is suitable for large graphs.

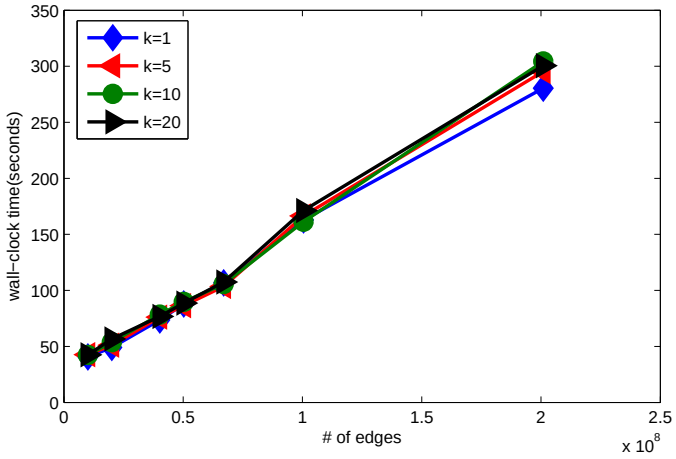
7.7 Related Work

In this section, we review the related work, which can be categorized into 2 parts: measuring the importance of nodes on graphs and spectral graph analysis. For the related work on general graph

[7]Another possible heuristic is to delete one node with the highest eigen-score $\mathbf{u}(i)$ from the *current* graph; and we repeat this procedure k times. But this method is still k times slower than NetShield and it is not clear how close the resulting solution is to the optimal solution.



(a) changing n (fix $m = 119,460$)



(b) changing m (fix $n = 2,667,119$)

Figure 7.16: *Evaluation of the scalability of the proposed NetShield wrt.* n *and* m*, respectively. Our NetShield scales linearly wrt* n *and* m*.*

mining, please refer to Chapter 6.

Measuring Importance of Nodes on Graphs. In the literature, there are a lot of node importance measurements, including betweenness centrality, both the one based on the shortest path [Fre77] and the one based on random walk [New05], PageRank [PBMW98], HITS [Kle98]. A remotely related work is the abnormality score of a given node [SQCF05]. Our 'Backboneness' score is *fundamentally* different from these node importance scores, in the sense that they *all* aim to measure the importance of an individual node; whereas our 'Backboneness' tries to *collectively* measure the importance of a set of k nodes. Even in the special case of $k = 1$, the existing node importance measurements all have subtle issues and occasionally disagree with intuitions, as we showed in Section 7.4, despite the fact that all these measures are successful for the goal they were

originally designed for. Moreover, several of these importance measurements do not scale up well for large graphs, being cubic or quadratic wrt the number of nodes n, even if we use approximations (e.g., [MW08]). In contrast, the proposed NetShield is linear wrt the number of edges and the number of nodes ($O(nk^2 + m)$).

Spectral Graph Analysis. Pioneering works in this aspect can be traced back to Fiedler's seminar work [Fie73]. Since then, spectral graph analysis has been a very hot research topic. Representative works include [SM97, NJW01, ZHD+01, DLJ08], etc. All of these works use the eigen-vectors of the graph (or the graph Laplacian) to find communities in the graph. In contrast, relatively less works explore the strength of the spectrum (i.e., eigenvalues) in graph mining. The two related works which motivate the adoption of λ as 'Vulnerability' measure are (1) epidemic threshold τ of a graph [CWW+07], where under a flu-like epidemic model, the authors in [CWW+07] show that τ is only determined by the first eigen-value of the adjacency matrix. and (2) triangle counting [Tso08], where the authors shows that the numbers of the triangles in the graph is totally determined by its spectrum. The proposed 'Vulnerability' measure of the graph also relates to the second smallest eigenvalue of graph Laplacian (known as graph algebra connectivity). There are two reasons why we eventually do not use graph algebra connectivity as our 'Vulnerability' measure: (1) implicitly, the second smallest eigenvalue of graph Laplacian measures the separability of the graph if we assume there are two communities in the graph (i.e., how these two communities are connected with each other); whereas λ of the adjacency matrix does not have such an assumption; (2) computationally, it is unclear whether or not we can develop similar scalable algorithms (i.e., linear wrt the size of the graph) to find a subset of nodes whose absence creates the maximum change of graph algebra connectivity.

7.8 Conclusion

We studied the 'Vulnerability' of large real graphs in this chapter. Our main contributions are

1. A novel 'Vulnerability' measurement (λ) for the graph, motivated from immunology and graph loop capacity;
2. A novel definition of 'Backboneness' score Br($\mathcal{S}$) for a set of nodes, by carefully using the results from the theory of matrix perturbation;
4. A *near-optimal* and *scalable* algorithm (NetShield) to find a set of nodes with the highest 'Backboneness' score, by carefully using the results from the theory of sub-modularity.
5. Justifications, proofs and complexity analysis, showing the intuitions, accuracy and efficiency of the proposed methods.
6. Extensive experiments on several real data sets, showing the effectiveness and efficiency of the proposed methods. For the effectiveness, our methods (1) lead to an effective immunization strategy, and (2) always give the mining results which are consistent with human intuitions. For the efficiency, our algorithm (1) achieves significant speed-up over straightforward solutions (*up to 7 orders of magnitude speedup*); and (2) is scalable for large graphs (*linear* wrt the size of the graph).

A promising research direction is to parallelize the current method (e.g., using Hadoop).

Part V

Mining Dynamic Graphs

Chapter 8

Anomaly Detection

Summary of This Chapter

- **Questions we want to answer:**

  Q: Given a large graph, how to summarize it and find anomalies?

- **Our answers and contributions**

  A: We proposed a family of novel, low rank approximation methods for static and dynamic graphs, which is provably equal or better compared with the best known methods in the literature, with the same accuracy.

8.1 Introduction

Graphs appear in a wide range of settings, like computer networks, the world wide web, biological networks, social networks and many more. How can we find patterns, e.g. communities and anomalies, in a large sparse graph? How can we track such patterns of interest if the graph is evolving over time?

A common representation of a graph is a matrix, such as an adjacency matrix for a unipartite graph where every row/column corresponds to a node in the graph, and every non-zero entry is an edge; an interaction matrix for a bipartite graph where rows and columns correspond to two different types of nodes and non-zero entries denote edges between them.

Naturally, low-rank approximations on matrices provide powerful tools to answer the above questions. Formally, a rank-c approximation of matrix $\mathbf{A}$ is a matrix $\tilde{\mathbf{A}}$ where $\tilde{\mathbf{A}}$ is of rank c and $\|\tilde{\mathbf{A}} - \mathbf{A}\|$ is small. The low-rank approximation is usually presented in a factorized form e.g., $\tilde{\mathbf{A}} = \mathbf{LMR}$ where $\mathbf{L}$, $\mathbf{M}$, and $\mathbf{R}$ are of rank-c.

Depending on the properties of those matrices, many different approximations have been proposed in the literature. For example, in SVD [GVL89], $\mathbf{L}$ and $\mathbf{R}$ are orthogonal matrices whose columns/rows are singular vectors and $\mathbf{M}$ is a diagonal matrix whose diagonal entries are singular values. Among all the possible rank-c approximations, SVD gives the best approximation in terms

of squared error. However, the SVD is usually dense, even if the original matrix is sparse. Furthermore, the singular vectors are abstract notions of best orthonormal basis, which is not intuitive for the interpretation.

Recently, alternatives have started to appear, such as CUR [DKM05b] and CMD [SXZF07], which use the actual columns and rows of the matrix to form $\mathbf{L}$ and $\mathbf{R}$. We call these *example-based low-rank approximations*. The benefit is that they provide an intuitive as well as sparse representation, since $\mathbf{L}$ and $\mathbf{R}$ are directly sampled from the original matrix. However, the approximation is often sub-optimal compared to SVD and the matrix $\mathbf{M}$ is no longer diagonal, which means a more complicated interaction.

Despite of the vast amount of literature on these topics, one of the major research challenges lies in the efficiency: (1) for a static graph, given the desired approximation accuracy, we want to compute the example-based low-rank approximation with the least computational and space cost; and (2) for a dynamic graph[1], we want to monitor/track this approximation efficiently over time.

To deal with the above challenges, we propose the family of *Colibri* methods. Adjacency matrices for large graphs may contain near-duplicate columns. For example, all nodes that belong to the same closed and tightly-connected community would have the same sets of neighbors (namely, the community's members). CMD addresses the problem of duplicate elimination. However, even without duplicates, it is still possible that the columns of $\mathbf{L}$ are linearly dependent, leading to a redundant representation of the approximating subspace, which wastes both time and space. The main idea of our method for static graphs (*Colibri-S*) is to eliminate linearly dependent columns while iterating over sampled columns to construct the subspace used for low rank approximation. Formally, the approximation $\tilde{\mathbf{A}} = \mathbf{LMR}$ where $\mathbf{L}$ consists of judiciously selected columns, $\mathbf{M}$ is an incrementally maintained core matrix, and $\mathbf{R}$ is another small matrix. *Colibri-S* is provably better or equal compared to the best competitors in the literature, in terms of both speed and space cost, while it achieves the same approximation accuracy. In addition, we provide an analysis of the gains in terms of the redundancy present in the data. Furthermore, our experiments on real data show significant gains in practice. With the same approximation accuracy, *Colibri-S* is up to $52\times$ faster than the best known competitor, while it only requires about 1/3 of the space.

For dynamic graphs, we propose *Colibri-D*. Again, for the same accuracy, *Colibri-D* is provably better or equal compared to the best known methods (including our own *Colibri-S*) in terms of speed. The main idea of *Colibri-D* is to leverage the "smoothness", or similarity between two consecutive time steps, to quickly update the approximating subspace. Our experiments show that, with the same accuracy, *Colibri-D* achieves up to $112\times$ speedup over the best published competitor, and is 5 times faster than *Colibri-S* applied from scratch for each time step.

The main contributions of this chapter are summarized as follows:

- A family of novel, low rank approximation methods (*Colibri-S*, *Colibri-D*) for static and dynamic graphs, respectively;
- Proofs, and complexity analysis, showing our methods are provably equal or better compared to the best known methods in the literature, for the same accuracy;
- Extensive experimental evaluation, showing that our methods are significantly faster, and

[1] In this paper, we use 'dynamic graphs' and 'time-evolving graphs' interchangeably.

nimbler than the top competitors. See Figure 8.1 for an example of the time and space savings of our *Colibri-S* over CUR and CMD [SXZF07].

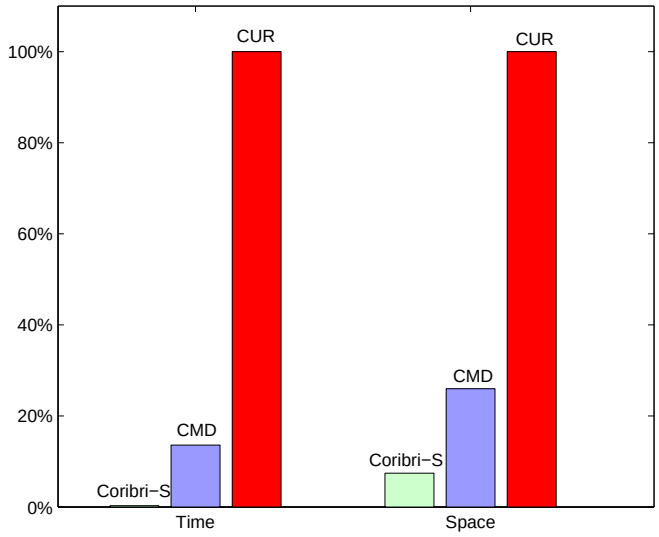


Figure 8.1: *Colibri-S* is significantly more efficient than both CUR and CMD in terms of both speed and space. Note that all these methods lead to the same approximation accuracy. Both speed and space cost are normalized by the most expensive one (i.e., CUR in both cases).

The rest of this chapter is organized as follows: we introduce notation and formally define the problems in Section 8.2. We present and analyze the proposed *Colibri-S* and *Colibri-D* in Section 8.3.3 and Section 8.4.2, respectively. We perform some case studies in Section 8.5.2 and provide experimental evaluation in Section 8.6.3. We review the related work in Section 8.7. Finally, we conclude in Section 8.8.

8.2 Problem Definitions

Table 8.2 lists the main symbols we use throughout the chapter. In this chapter, we consider the case of bipartite graphs. Uni-partite graph can be viewed as a special case. We represent a general bipartite graph by its adjacency matrix[2]. Following the standard notation, we use capital letters for matrices (e.g. $\mathbf{A}$), arrows for vectors (e.g. $\vec{a}_j$), and calligraphic fonts for sets (e.g. $\mathcal{I}$). We denote the transpose with a prime (i.e., $\mathbf{A}'$ is the transpose of $\mathbf{A}$), and we use parenthesized superscripts to denote time (e.g., $\mathbf{A}^{(t)}$ 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. $\mathbf{A}_{n\times l}$ means a matrix of size $n \times l$). Also, we represent the elements in a matrix using a convention similar to Matlab, e.g., $\mathbf{A}(\mathbf{i}, \mathbf{j})$ is the element at the i^{th} row

[2]In practice, we store these matrices using an adjacency list representation, since real graphs are often very sparse.

Table 8.1: Symbols

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}, \ldots$	matrices (bold upper case)
$\mathbf{A}(i, j)$	the element at the i^{th} row and j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}(i, :)$	the i^{th} row of matrix $\mathbf{A}$
$\mathbf{A}(:, j)$	the j^{th} column of matrix $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\vec{a}, \vec{b}, \ldots$	column vectors
$\mathcal{I}, \mathcal{J}, \ldots$	sets (calligraphic)
$\mathbf{A}^{(t)}$	$n \times l$ time-aggregate interaction matrix at time t
$\vec{a}_j^{(t)}$	the j^{th} column of $\mathbf{A}^{(t)}$, i.e., $\vec{a}_j^{(t)} = \mathbf{A}^{(t)}(:, j)$
$\mathcal{I}$	indices for columns sampled: $\mathcal{I} = \{i_1, ..., i_c\}$
n, l	number of for type 1 and type 2 objects, respectively
c	sample size. i.e. the number of columns sampled
$\mathbf{C}_0^{(t)}$	$n \times c$ initial sampling matrix, consisting of c columns from $\mathbf{A}^{(t)}$. i.e., $\mathbf{C}_0^{(t)} = \mathbf{A}^{(t)}(:, \mathcal{I})$
$m_0^{(t)}$	number of edges in $\mathbf{C}_0^{(t)}$ at time t

and j^{th} column of the matrix $\mathbf{A}$, and $\mathbf{A}(:, \mathbf{j})$ is the j^{th} column of $\mathbf{A}$, etc. With this notation, we can define matrix $\mathbf{C}_0$ as $\mathbf{C}_0 = \mathbf{A}(:, \mathcal{I}) = [\mathbf{A}(:, i_1), ..., \mathbf{A}(:, i_c)]$. In other words, $\mathbf{C}_0$ is the sub-matrix of $\mathbf{A}$ by stacking all its columns indexed by the set $\mathcal{I}$.Without loss of generality, we assume that the numbers of type 1 and type 2 objects (corresponding to rows and columns in the adjacency matrix) 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, with associated edge weights. While there are multiple choices to update the adjacency matrix (e.g. sliding window, exponential forgetting etc), we use global aggregation for simplicity: once an edge appears at some time step t, the corresponding entry of the adjacency matrix is updated and the edge is never deleted or modified. This assumption facilitates presentation, but our methods can naturally apply to other update schemes.

With the above notations and assumptions, our problems can be formally defined as follows:

Problem 10. (Static Case.) *Low rank approximation for static sparse graphs*

Given: *A large, static sparse graph $\mathbf{A}_{n \times l}$, and sample size c;*

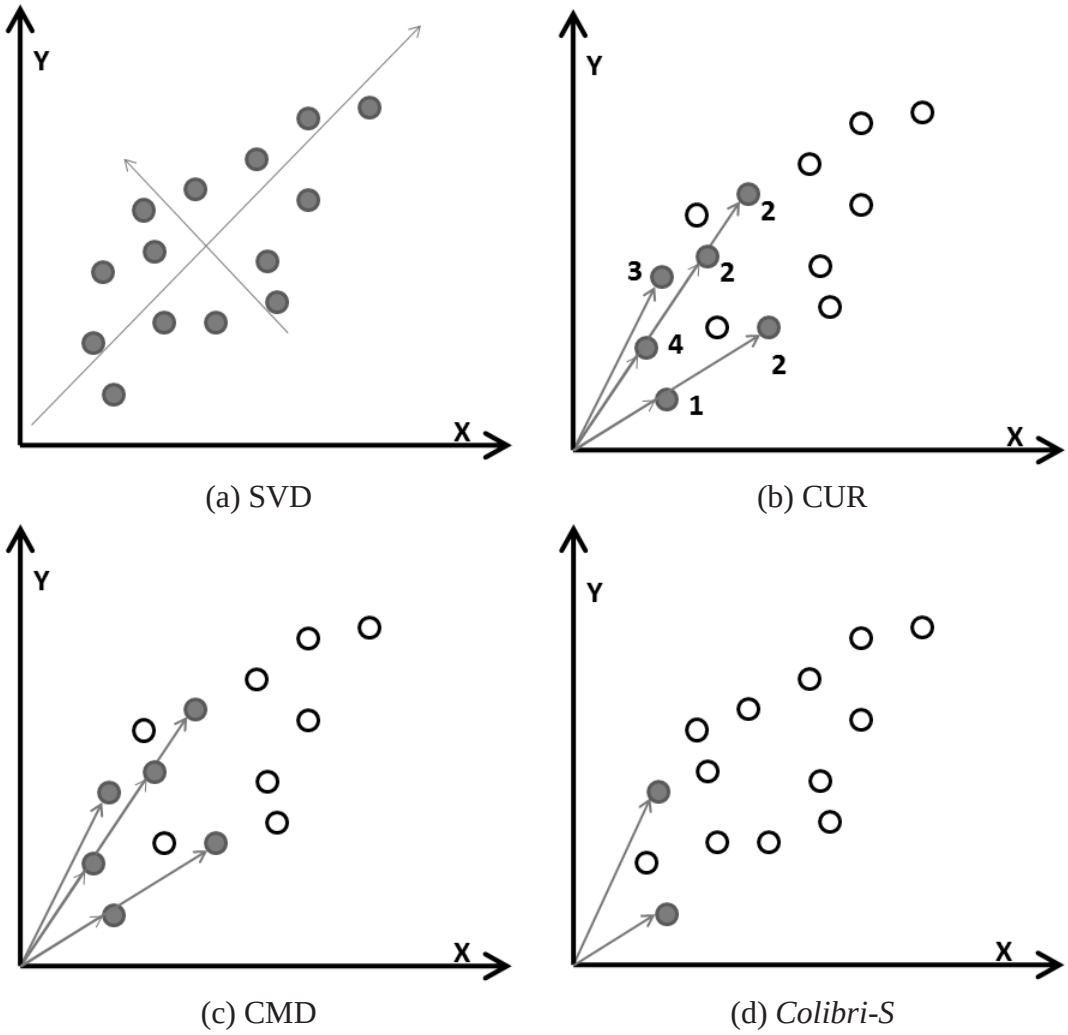
Find: *Its low-rank approximation structure efficiently. That is, find three matrices $\mathbf{L}_{n \times \tilde{c}}, \mathbf{M}_{\tilde{c} \times \tilde{c}}$, and $\mathbf{R}_{\tilde{c} \times l}$ such that $\mathbf{A}_{n \times l} \approx \mathbf{L}_{n \times \tilde{c}} \mathbf{M}_{\tilde{c} \times \tilde{c}} \mathbf{R}_{\tilde{c} \times l}$, where $\tilde{c} \leq c$.*

Problem 11. (Dynamic Case.) *Low rank approximation for dynamic sparse graphs*

Given: *A large, dynamic sparse graph $\mathbf{A}_{n \times l}^{(t)}$, for $t = 1, 2, \ldots$, and the sample size c;*

Track: *Its low-rank approximation structure over time efficiently. That is, to find three matrices $\mathbf{L}^{(t)}, \mathbf{M}^{(t)}$, and $\mathbf{R}^{(t)}$ for each time step t such that $\mathbf{A}_{n \times l}^{(t)} \approx \mathbf{L}_{n \times \tilde{c}^{(t)}}^{(t)} \mathbf{M}_{\tilde{c}^{(t)} \times \tilde{c}^{(t)}}^{(t)} \mathbf{R}_{\tilde{c}^{(t)} \times l}^{(t)}$, where $\tilde{c}^{(t)} \leq c$.*

8.3 *Colibri-S* for Static Graphs



(a) SVD (b) CUR

(c) CMD (d) *Colibri-S*

Figure 8.2: A pictorial comparison for different methods. To construct the same subspace, SVD will use all the data points (dark ones); CUR will use a subset of data point with possibly a lot duplications (the number besides the arrow is the number of duplicate copies); CMD will remove the duplicate the columns in CUR; and our *Colibri-S* will further remove all linearly dependent columns which is most efficient in both speed and space. For illustrative purpose, we set the approximation accuracy of each method to be always 100% in this example.

In this section, we address Problem 10 and introduce our *Colibri-S* for static graphs. After some necessary background in subsection 8.3.1, we present the algorithm in subsection 8.3.2, followed by the proofs and complexity analysis in subsection 8.3.3.

8.3.1 Preliminaries

Here, we want to decompose the adjacency matrix $\mathbf{A}_{n\times l}$ of a static graph into three matrices: $\mathbf{L}_{n\times\tilde{c}}$, $\mathbf{M}_{\tilde{c}\times\tilde{c}}$, and $\mathbf{R}_{\tilde{c}\times l}$. The goal is to achieve a good balance between efficiency and approximation quality. For the quality, we want $\tilde{\mathbf{A}} = \mathbf{LMR}$ to approximate the original adjacency matrix $\mathbf{A}$ as well as possible. Throughout the paper, we use the Frobenius norm of $\tilde{\mathbf{A}} - \mathbf{A}$ to measure the approximation error. As for efficiency, we want to (1) keep the matrices $\mathbf{L}$ and $\mathbf{R}$ small ($\tilde{c} \ll l$) and sparse, to save space; and (2) compute the decomposition using minimal running time.

The best known methods to achieve such balance are CUR [DKM05b] and its improved version, CMD [SXZF07]. The key idea behind CUR and CMD is to sample some columns of $\mathbf{A}$ with replacement, biased towards those with larger norms[3]; and then to use the projection of the original adjacency matrix $\mathbf{A}$ into the subspace spanned by these sampled columns as the low rank approximation of the matrix $\mathbf{A}$. As shown in [DKM05b], such procedures provably achieve an optimal approximation. Additionally, the matrices $\mathbf{L}$ and $\mathbf{R}$ by CUR/CMD are usually very sparse, thus the CUR/CMD decomposition is shown to be much faster than standard SVD.

8.3.2 Algorithm

Our algorithm shares the same high-level principle as CUR and CMD. That is, we want to sample some columns of the matrix $\mathbf{A}$ and then project $\mathbf{A}$ into the subspace spanned by these columns. As we show later, our method achieves exactly the same approximation accuracy as CUR/CMD, but it is equal or better compared to CUR/CMD in terms of both space and time.

If we concatenate all the sampled columns into a matrix $\mathbf{C}_0$, we can use $\mathbf{C}_0(\mathbf{C}_0'\mathbf{C}_0)^\dagger\mathbf{C}_0'\mathbf{A}$ as the approximation of the original adjacency matrix $\mathbf{A}$, where $(\mathbf{C}_0'\mathbf{C}_0)^\dagger$ is the Moore-Penrose pseudo-inverse of the square matrix $\mathbf{C}_0'\mathbf{C}_0$.

However, the sampled columns in $\mathbf{C}_0$ may contain duplicates (or near duplicates)—for example, all nodes that belong to the same closed and tightly-connected community would have the same sets of neighbors (namely, the community's members). CMD essentially performs duplicate elimination. However, more generally, the columns of $\mathbf{C}_0$ may be unequal but linear dependences may still be present. In other words, the columns of $\mathbf{C}_0$ form a *redundant* or *overcomplete* basis. This is clearly not efficient in terms of space. Moreover, if we keep these redundant columns, we have to estimate the pseudo-inverse of a larger matrix, which adversely affects running time as well.

The heart of *Colibri-S* is to iteratively construct the desired subspace, eliminate these redundant columns in the process. Algorithm 14 shows the full pseudocode.

There are three stages in algorithm 14. First (steps 1-2), we sample c columns of matrix $\mathbf{A}$ with replacement, biased towards those with higher norms, exactly as CUR does (first step in Figure. 8.3). Then, we try to select linearly independent columns from the initially sampled columns and build the $\mathbf{M}$ matrix (referred to as the "core matrix"): after an initialization step (step 3), we iteratively test if a new column $\mathbf{A}(:, i_k)$ is linearly dependent on the current columns of $\mathbf{L}$ (steps 5-7). If so, we skip the column $\mathbf{A}(:, i_k)$. Otherwise, we append $\mathbf{A}(:, i_k)$ into $\mathbf{L}$ and update the

[3] In [DKM05b, SXZF07], the authors also suggest simultaneously sampling columns and rows. Our method can be naturally generalized to handle this case. For simplicity, we focus on sampling columns only.

Algorithm 14 *Colibri-S* for Static Graphs

Require: The adjacency matrix $\mathbf{A}_{n\times l}$, tolerance ϵ, and the sample size c
Ensure: Three matrices $\mathbf{L}_{n\times\tilde{c}}$, $\mathbf{M}_{\tilde{c}\times\tilde{c}}$, and $\mathbf{R}_{\tilde{c}\times l}$, where $\tilde{c} \leq c$.

1: Compute column distribution for $x = 1, ..., l$: $P(x) = \sum_i \mathbf{A}(i,x)^2 / \sum_{i,j} \mathbf{A}(i,j)^2$;
2: Sample c columns from $\mathbf{A}$ based on $P(x)$. Let $\mathcal{I} = \{i_1, ..., i_c\}$ be the indices of these columns.
3: Initialize $\mathbf{L} = [\mathbf{A}(:,i_1)]$; $\mathbf{M} = 1/(\mathbf{A}(:,i_1)' \cdot \mathbf{A}(:,i_1))$
4: **for** $k = 2 : c$ **do**
5: Compute the residual: $r\vec{e}s = \mathbf{A}(:,i_k) - \mathbf{LML}'\mathbf{A}(:,i_k)$
6: **if** $\|r\vec{e}s\| \leq \varepsilon\|\mathbf{A}(:,i_k)\|$ **then**
7: Continue;
8: **else**
9: Compute: $\delta = \|r\vec{e}s\|^2$; and $\vec{y} = \mathbf{ML}'\mathbf{A}(:,i_k)$
10: Update the core matrix $\mathbf{M}$: $\mathbf{M} \leftarrow \begin{pmatrix} \mathbf{M} + \vec{y}\vec{y}/\delta & -\vec{y}/\delta \\ -\vec{y}'/\delta & 1/\delta \end{pmatrix}$
11: Expand $\mathbf{L}$: $\mathbf{L} \leftarrow [\mathbf{L},\ \mathbf{A}(:,i_k)]$
12: **end if**
13: **end for**
14: Compute $\mathbf{R} = \mathbf{L}'\mathbf{A}$.

core matrix $\mathbf{M}$ (steps 9-11). Note that if the new column $\mathbf{A}(:,i_k)$ is linearly independent wrt the current columns in $\mathbf{L}$ (i.e., if $\|r\vec{e}s\| > \varepsilon\|\mathbf{A}(:,i_k)\|$), we can use the residual $r\vec{e}s$ computed in step 5 to update the core matrix $\mathbf{M}$ in step 9. Conversely, we use the core matrix $\mathbf{M}$ to estimate the residual and test linear dependence of the new column (step 5). In this way, we simultaneously prune the redundant columns and update the core matrix. The last step in Figure. 8.3 shows the final $\mathbf{L}$ obtained after eliminating the redundant columns from $\mathbf{C}_0$. Finally, we define the $\mathbf{R}$ matrix to be $\mathbf{L}'\mathbf{A}$.[4]

8.3.3 Proofs and Analysis

Here we provide the proofs and the performance analysis of *Colibri-S*. We also make a brief comparison with the state-of-art techniques, such as CUR/CMD.

Proof of Correctness for *Colibri-S*

We have the following theorem for the correctness of Alg. 14:

Theorem 6. Correctness of *Colibri-S*. *Let the matrix $\mathbf{C}_0$ contain the initial sampled columns from $\mathbf{A}$(i.e. $\mathbf{C}_0 = \mathbf{A}(:,\mathcal{I})$). With tolerance $\epsilon = 0$, the following facts hold for the matrices $\mathbf{L}$ and $\mathbf{M}$ in Alg. 14:*

[4]Note that while $\mathbf{L}$ is sparse since it consists of a subset of the original columns from $\mathbf{A}$, the matrix $\mathbf{R}$ is the multiplication of two sparse matrices and is not necessarily sparse. In order to further save space, we can use a randomized algorithm [DKM05a] to approximate $\mathbf{R}$. This can be naturally incorporated into Alg. 14. However, it is an orthogonal to what we are proposing in this paper. For simplicity, we will use $\mathbf{R} = \mathbf{L}'\mathbf{A}$ throughout this paper.

P1: the columns of $\mathbf{L}$ *are linearly independent;*
P2: $\mathbf{L}$ *shares the same column space as* $\mathbf{C}_0$*;*
P3: the core matrix $\mathbf{M}$ *satisfies* $\mathbf{M} = (\mathbf{L}'\mathbf{L})^{-1}$.

Proof. First, we will prove 'P3' in Theorem 6 by induction. The base case (step 3 of Alg. 14) is obviously true.

For the induction step of 'P3', let us suppose that (1) $\mathbf{M} = (\mathbf{L}'\mathbf{L})^{-1}$ holds up to the $k_1^{th} (2 \leq k_1 \leq c)$ iteration; and (2) $\mathbf{L}$ will be expanded next in the k_2^{th} iteration ($k_1 < k_2 \leq c$).

Let $\tilde{\mathbf{L}} = (\mathbf{L} \;\; \mathbf{A}(:, i_{k_2}))$. We have

$$\begin{aligned} \tilde{\mathbf{L}}'\tilde{\mathbf{L}} &= \begin{pmatrix} \mathbf{L}' \\ \mathbf{A}(:, i_{k_2})' \end{pmatrix} \times \begin{pmatrix} \mathbf{L} & \mathbf{A}(:, i_{k_2}) \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{L}'\mathbf{L} & \mathbf{L}'\mathbf{A}(:, i_{k_2}) \\ \mathbf{A}(:, i_{k_2})'\mathbf{L} & \mathbf{A}(:, i_{k_2})'\mathbf{A}(:, i_{k_2}) \end{pmatrix} \end{aligned} \tag{8.1}$$

Define $\tilde{\mathbf{M}} = \begin{pmatrix} \mathbf{M} + \vec{y}'\vec{y}/\delta & -\vec{y}/\delta \\ -\vec{y}'/\delta & 1/\delta \end{pmatrix}$, where $\vec{y}$ and δ are defined in Alg. 14.

Since $\mathbf{M} = (\mathbf{L}'\mathbf{L})^{-1}$ by inductive hypothesis, it can be verified that $r\vec{e}s$ is the residual if we project the column $\mathbf{A}(:, i_{k_2})$ into the column space of $\mathbf{L}$. Based on the orthogonality property of the projection, we have

$$\begin{aligned} \delta &= \|r\vec{e}s\|^2 \\ &= r\vec{e}s'(r\vec{e}s + \mathbf{L}\mathbf{M}\mathbf{L}'\mathbf{A}(:, i_{k_2})) \\ &= r\vec{e}s'\mathbf{A}(:, i_k) \end{aligned} \tag{8.2}$$

Now, applying the Sherman-Morrison lemma [PC90] to the matrix $\tilde{\mathbf{L}}'\tilde{\mathbf{L}}$ in the form of eq. 8.1, based on eq. 8.2, we can verify that $\tilde{\mathbf{M}} = (\tilde{\mathbf{L}}'\tilde{\mathbf{L}})^{-1}$ holds, which completes the proof of 'P3'.

Next, let us prove 'P1' in Theorem 6 by induction. Again, the base case for 'P1' is obviously true (step 3 of Alg. 14).

For the induction step for 'P1', let us suppose that (1) all the columns in $\mathbf{L}_{n \times \hat{c}}$ are linearly independent up to the k_1^{th} iteration $(2 \leq \hat{c} \leq k_1 \leq c)$; and (2) $\mathbf{L}$ will be expanded next in the k_2^{th} iteration ($k_1 < k_2 \leq c$). We only need to prove that $\mathbf{A}(:, i_{k_2}))$ is linear independent wrt the columns in the current $\mathbf{L}$ matrix.

By 'P3', the $r\vec{e}s$ computed in step 5 is the exactly the residual if we project the column $\mathbf{A}(:, i_{k_2}))$ into the column space spanned by the current $\mathbf{L}$ matrix. Since we decide to expand $\mathbf{L}$ by $\mathbf{A}(:, i_{k_2}))$, with tolerance $\epsilon = 0$, it must be true that the residual satisfies $\mathrm{res} > 0$ (step 8). In other words, the column $\mathbf{A}(:, i_{k_2}))$ is not in the column space of $\mathbf{L}$.

Now, suppose that $\mathbf{A}(:, i_{k_2}))$ is linearly dependent to the columns in the current $\mathbf{L}$ matrix. The column $\mathbf{A}(:, i_{k_2}))$ must lie in the column space of $\mathbf{L}$. This is contra-positive, which completes the proof of 'P1'.

Finally, from 'P1', for each column $\vec{u} \in \{\mathbf{C_0} - \mathbf{L}\}$ (steps 5-7 of Alg. 14), there must exist a vector $\vec{\beta} = (\beta_1, ..., \beta_{\tilde{c}})' = \mathbf{M}\mathbf{L}'\vec{u}$, such that $\vec{u} = \mathbf{L}\vec{\beta}$ holds. In other words, $\vec{u}$ must be in the column space of $\mathbf{L}$. Therefore, removing the column $\vec{u}$ from $\mathbf{L}$ will not change the column space of $\mathbf{L}$. This completes the proof of 'P2'. □

Notice that *Colibri-S* iteratively finds the linearly independent set of columns (i.e., the matrix $\mathbf{L}$). For the same initially sampled columns ($\mathbf{C}_0$), it might lead to a different $\mathbf{L}$ matrix if we use a different order in the index set $\mathcal{I}$. However, based on Theorem 1, this operation will not affect the subspace spanned by the columns of the matrix $\mathbf{L}$ since it is always the same as the subspace spanned by the columns of the matrix $\mathbf{C}_0$. Therefore, it will not affect the approximation accuracy for the original matrix $\mathbf{A}$.

Efficiency of *Colibri-S*

We have the following lemma for the speed of Alg. 14.

Lemma 17. Efficiency of *Colibri-S*. *The computational complexity to output* $\mathbf{M}$ *and* $\mathbf{L}$ *in Alg. 14 is bounded by* $O(c\tilde{c}^2 + c\tilde{m})$*, where* $\tilde{c}, \tilde{m}$ *are the number of columns and edges in the matrix* $\mathbf{L}$*, respectively; and* c *is the number of columns in* $\mathbf{C}_0$*.*

Proof. In the k^{th} iteration of Alg. 14, suppose there are $\hat{k}$ columns and $\hat{m}$ edges in the matrix $\mathbf{L}$. We have $\hat{k} \leq k$ and $\hat{m} \leq \tilde{m}$.

We assume that $\mathbf{L}$ and $\mathbf{A}$ are stored as adjacency lists, since they are sparse, and $\mathbf{M}$ is stored as a full matrix, since it is usually dense. With this storage format, it is easy to verify that the cost of the k^{th} iteration (ignoring constant factors) of Alg. 14 is $\hat{k}^2 + \hat{m}$.

Let us first consider the time cost for all these $\tilde{c}$ columns in $\mathbf{L}$. Notice that each time we expand one such column, the size of $\mathbf{M}$ will increase by exactly 1×1. Therefore, the total running time (again, ignoring factors) for expanding these columns in Alg. 14 is:

$$\begin{aligned} \text{time}_1 &= \sum_{k=2, \mathbf{A}(:,i_k) \in \mathbf{L}}^{c} (\hat{k}^2 + \hat{m}) \\ &\leq \sum_{i=1}^{\tilde{c}-1} (i^2 + \tilde{m}) \\ &\leq \sum_{i=1}^{\tilde{c}-1} (i^2) + \tilde{c}\tilde{m}) \\ &= O(\tilde{c}^3 + \tilde{c}\tilde{m}) \end{aligned} \tag{8.3}$$

Next, let us consider the time cost for all these $(c - \tilde{c})$ redundant columns. For each of those columns, we have $\hat{k} \leq \tilde{c}$ and $\hat{m} \leq \tilde{m}$. Therefore, the total running time for eliminating these $(c-\tilde{c})$ columns is:

$$\begin{aligned} \text{time}_2 &= \sum_{k=2, \mathbf{A}(:,i_k) \in \{\mathbf{C}_0 - \mathbf{L}\}}^{c} (\hat{k}^2 + \hat{m}) \\ &\leq \sum_{i=1}^{c-\tilde{c}} (\tilde{c}^2 + \tilde{m}) \\ &= O(c\tilde{c}^2 - \tilde{c}^3 + (c - \tilde{c})\tilde{m}) \end{aligned} \tag{8.4}$$

Putting eq. 8.3 and 8.4 together, we get that the total running time for steps 4–13 of Alg. 14 is:

$$\begin{aligned} \text{time} &= \text{time}_1 + \text{time}_2 \\ &= O(c\tilde{c}^2 + c\tilde{m}) \end{aligned} \tag{8.5}$$

which completes the proof of Lemma 17. □

Comparison with CUR/CMD

Next we compare *Colibri-S* against the state-of-art techniques, i.e. CUR [DKM05b] and CMD [SXZF07]. We compare with respect to accuracy, time and space cost.

Lemma 18 (ACCURACY)**.** *Using the same initial sampled columns* $\mathbf{C}_0$*, Alg. 14 has exactly the same approximation accuracy as CUR [DKM05b] and CMD [SXZF07].*

Proof. Define $\tilde{\mathbf{A}}$ as $\tilde{\mathbf{A}} = \mathbf{LMR}$. By Theorem 6, the matrix $\tilde{\mathbf{A}}$ satisfies $\tilde{\mathbf{A}} = \mathbf{L}(\mathbf{L}'\mathbf{L})^{-1}\mathbf{L}'\mathbf{A}$. In other words, $\tilde{\mathbf{A}}$ is the projection of the matrix $\mathbf{A}$ into the column space of $\mathbf{L}$. On the other hand, by Theorem 6, the matrix $\mathbf{L}$ has the same column space as $\mathbf{C}_0$, i.e., $\tilde{\mathbf{A}} = \mathbf{C}_0(\mathbf{C}_0'\mathbf{C}_0)^{\dagger}\mathbf{C}_0'\mathbf{A}$, which is exactly how CUR/CMD [DKM05b, SXZF07] tries to approximate the original matrix $\mathbf{A}$. □

Lemma 19 (SPACE)**.** *Using the same initial sampled columns* $\mathbf{C}_0$*, Alg. 14 is better than or equal to CUR in [DKM05b] and CMD in [SXZF07] in terms of space.*

Proof. Notice that $\mathbf{L}$ is always a subset of $\mathbf{C}_0$. On the other hand, if there exist duplicate columns in $\mathbf{C}_0$, they will appear only once in $\mathbf{L}$. □

Lemma 20 (TIME)**.** *Using the same initial sampled columns* $\mathbf{C}_0$*, Alg. 14 is faster than, or equal to CUR ([DKM05b]) and CMD ([SXZF07]).*

Proof. By Lemma 17, the computational complexity of Alg. 14 at the worst case is the same as the original CUR method in [DKM05b] ($O(cm)$ for multiplying $\mathbf{C}_0'$ and $\mathbf{C}_0$ together; and $O(c^3)$ for the Moore-Penrose pseudo-inverse of $\mathbf{C}_0'\mathbf{C}_0$. Also notice that $\tilde{c} \leq c$ and $\tilde{m} \leq m$). On the other hand, if there exist duplicate columns in $\mathbf{C}_0$, we can always remove them before step 3 in Alg.14 and then CMD in [SXZF07] will degenerate to CUR [DKM05b]. □

In particular, the complexity is proportional to the square of the "true" dimensionality $\tilde{c}$ of the approximating subspace. Since, as we shall see in the experimental evaluation, in real datasets $\tilde{c}$ is significantly smaller than c, this translates to substantial savings in computation time as well as space.

INTUITION. The intuition behind the above proofs and savings is shown in Figure 8.2, which gives a pictorial comparison of our *Colibri-S* with SVD/CUR/CMD. Figure 8.2 shows that: (1) SVD (Figure 8.2(a)) uses all data points (dark ones) and the resulting $\mathbf{L}$ matrix is dense. (2) CUR (Figure 8.2(b)) uses sampled columns (dark ones) but there may be many duplicate columns (the number next to each arrow stands for the multiplicity) The resulting $\mathbf{L}$ matrix of CUR is sparse but it has totally 16 columns. (3) CMD (Figure 8.2(c)) removes the duplicate columns in CUR and the resulting $\mathbf{L}$ (with 6 columns) is more compact. (4) Our *Colibri-S* (Figure 8.2(d)) further removes

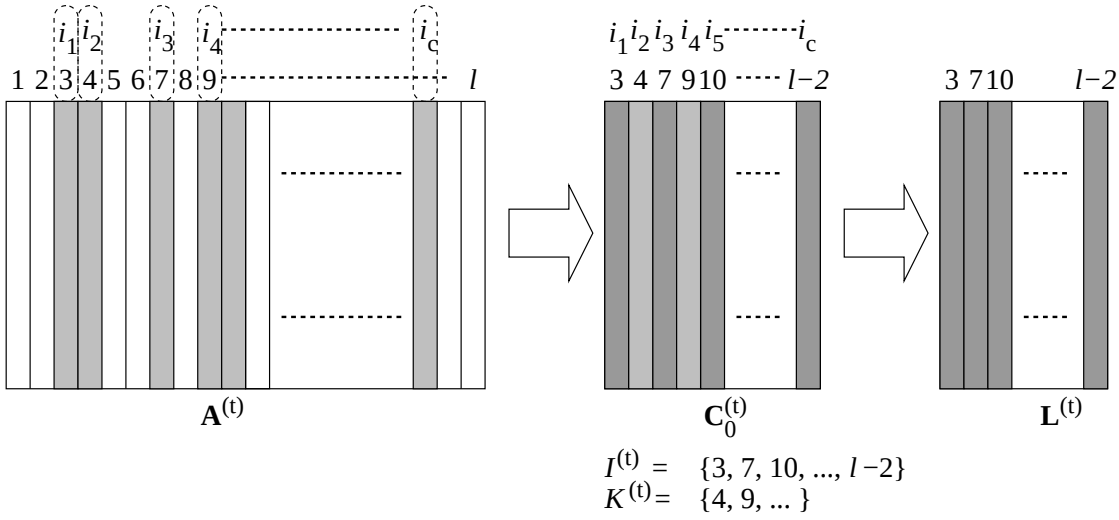


Figure 8.3: Illustration of notation and process for *Colibri-S*. Shaded columns are part of initial sample, dark shaded columns are linearly independent among those.

all the linearly dependent columns and the resulting $\mathbf{L}$ only contains 2 sparse columns. Therefore, while all these four methods leads to the same subspace, *Colibri-S* is most efficient in both time and space.

8.4 *Colibri-D* for Dynamic Graphs

In this section, we deal with Problem 11 and propose *Colibri-D* for dynamic, time-evolving graphs. Our goal is to find the low rank approximation structure of the adjacency matrix at each time step t efficiently. As for static graphs, we first give the algorithm in subsection 8.4.1 and then provide theoretical justification and analysis in subsection 8.4.2.

8.4.1 Algorithm

Conceptually, we could call Alg. 14 to output the low rank approximation for each time step t. In this way, we will have to compute the core matrix $\mathbf{M}$, which is the most expensive part in Alg. 14, for each time step from the scratch. On the other hand, if the graph changes "smoothly" between two consecutive time steps (i.e., the number of affected edges is reasonably small) then, intuitively, we do not expect its low rank approximation structure to change dramatically. This is exactly the heart of our *Colibri-D*. We want to leverage the core matrix $\mathbf{M}^{(t)}$ to quickly get the core matrix $\mathbf{M}^{(t+1)}$ in the next time step, given that the graph changes "smoothly" from time step t to $(t+1)$.

For simplicity, we assume that the indices of the initial sampled columns $\mathbf{C}_0^{(t)}$ are fixed. That is, we will fix the index set $\mathcal{I} = \{i_1, ..., i_c\}$ over time, and we will always use the projection of the adjacency matrix $\mathbf{A}^{(t)}$ in the columns space of $\mathbf{C}_0^{(t)} = \mathbf{A}^{(t)}(:, \mathcal{I})$ as the low rank approximation of

$\mathbf{A}^{(t)}$ for each time step[5]. Note that even if we use the same initial column indices, the content of matrix $\mathbf{C}_0^{(t)}$ keeps changing over time and so does the subspace it spans. Our goal is to efficiently update the non-redundant basis for the subspace spanned by the columns of $\mathbf{C}_0^{(t)}$ over time. Note that in Figure. 8.4, the column indices of $\mathbf{C}_0^{(t+1)}$ are exactly the same as those for $\mathbf{C}_0^{(t)}$ in Figure. 8.3. However, in this example, the contents of columns 3 and $l-2$ have changed.

The basic idea of our algorithm for dynamic graphs is as follows: once the adjacency matrix $\mathbf{A}^{(t+1)}$ at time step $(t+1)$ is updated, we will update the matrix $\mathbf{C}_0^{(t+1)}$. Then, we will try to identify those linearly independent columns $\mathbf{L}^{(t+1)}$ within $\mathbf{C}_0^{(t+1)}$ as well as the core matrix $\mathbf{M}^{(t+1)}$. To reduce the computational cost, we will leverage the core matrix from the current time step $\mathbf{M}^{(t)}$ to update $\mathbf{L}^{(t+1)}$ as well as $\mathbf{M}^{(t+1)}$, instead of computing them from the scratch. Finally, we will update the $\mathbf{R}$ matrix as $\mathbf{R}^{(t+1)} = \mathbf{L}^{(t+1)'}\mathbf{A}^{(t+1)}$.

Next, we will describe how to update $\mathbf{L}^{(t+1)}$ and $\mathbf{M}^{(t+1)}$ at time step $t+1$. At time step t, we might find some redundant columns in $\mathbf{C}_0^{(t)}$ which are linearly dependent wrt the remaining columns in $\mathbf{C}_0^{(t)}$. In Figure. 8.3, these were columns 4 and 9. We split the indices set $\mathcal{I}$ into two disjoint subsets: $\mathcal{J}^{(t)}$ and $\mathcal{K}^{(t)}$, as shown in Figure. 8.3. We require that $\mathcal{I} = \mathcal{J}^{(t)} \cup \mathcal{K}^{(t)}$, and $\mathbf{L}^{(t)} = \mathbf{A}^{(t)}(:, \mathcal{J}^{(t)})$. In other words, $\mathcal{J}^{(t)}$ corresponds to those columns in $\mathbf{C}_0^{(t)}$ that are actually used to construct the subspace; and $\mathcal{K}^{(t)}$ corresponds to those redundant columns in $\mathbf{C}_0^{(t)}$. Notice that even though we fix the index set $\mathcal{I}$ over time, the subsets $\mathcal{J}^{(t)}$ and $\mathcal{K}^{(t)}$ change over time. Updating the matrix $\mathbf{L}^{(t)}$ is equivalent to updating the subset $\mathcal{J}^{(t)}$. To simplify the description of the algorithm, we further partition $\mathcal{J}^{(t)}$ into two disjoint subsets $\mathcal{J}_a^{(t)}$ and $\mathcal{J}_b^{(t)}$, such that $\mathcal{J}^{(t)} = \mathcal{J}_a^{(t)} \cup \mathcal{J}_b^{(t)}$. We require that $\mathbf{A}^{(t)}(:, \mathcal{J}_a^{(t)}) = \mathbf{A}^{(t+1)}(:, \mathcal{J}_a^{(t)})$; and $\mathbf{A}^{(t)}(:, \mathcal{J}_b^{(t)}) \neq \mathbf{A}^{(t+1)}(:, \mathcal{J}_b^{(t)})$. In other words, $\mathcal{J}_a^{(t)}$ corresponds to those unchanged columns in $\mathbf{L}$ from t to $(t+1)$, while $\mathcal{J}_b^{(t)}$ corresponds to those changed columns from t to $(t+1)$. These sets are shown in Figure. 8.4 on the left: notice that their union is $\mathcal{I}^{(t)}$ from Figure. 8.3.

With the above notations, the complete pseudocode to update the low rank approximation from time step t to $(t+1)$ is given in Alg. 15.

Comparing Alg. 15 with its static version (Alg. 14), the main differences are (1) we do not need to test the linear dependence and build our core matrix from the scratch if the subset $\mathcal{J}_a$ is not empty (steps 3–9), since the columns in $\mathcal{J}_a$ are guaranteed to be linearly independent; (2) furthermore, if the change in $\mathcal{I}$ is relatively small (i.e. $\mid \mathcal{J}_a^{(t)} \mid > \mid \mathcal{J}_b^{(t)} \mid$), we do not need to initialize our core matrix $\mathbf{M}^{(t+1)}$ from the scratch. Instead, we can leverage the information in $\mathbf{M}^{(t)}$ to do fast initialization (steps 6–8). These strategies, as will be shown in the next subsection, will dramatically reduce the computational time, while the whole algorithm will give exactly the same low rank approximation as if we had called Alg. 14 for time step $(t+1)$. After we initialize the core matrix $\mathbf{M}^{(t+1)}$ (after step 9), we will recursively test the linear dependence for each column in $\mathcal{K}^{(t)}$ and $\mathcal{J}_b^{(t)}$ and possibly incorporate them to expand the core matrix $\mathbf{M}^{(t+1)}$, which is very similar to what we do for the static graphs in Alg. 14.

In our running example of Figure. 8.3 and 8.4, since columns 7 and 10 were linearly independent at time t and they have remained unchanged, we can safely initialize $\mathbf{L}^{(t+1)}$ to include these. However, since columns 3 and $l-2$ have changed, we need to re-test for linear independence. In

[5]How to update the indices set $\mathcal{I}$ over time is beyond the scope of this paper.

Algorithm 15 *Colibri-D* for Dynamic Graphs

Require: The adjacency matrices $\mathbf{A}^{(t)}$ and $\mathbf{A}^{(t+1)}$, the indices set $\mathcal{I} = \mathcal{J}^{(t)} \cup \mathcal{K}^{(t)}$, tolerance ϵ, and the core matrix $\mathbf{M}^{(t)}$ at time step t

Ensure: Three matrices $\mathbf{L}^{(t+1)}$, $\mathbf{M}^{(t+1)}$, and $\mathbf{R}^{(t+1)}$; and updated indices partition $\mathcal{I} = \mathcal{J}^{(t+1)} \cup \mathcal{K}^{(t+1)}$.

Set $\mathcal{J}_a^{(t)}$ and $\mathcal{J}_b^{(t)}$ based on $\mathbf{A}^{(t)}$ and $\mathbf{A}^{(t+1)}$;
Initialize $\mathbf{L}^{(t+1)} = \mathbf{A}(:, \mathcal{J}_a^{(t)})$; $\mathcal{K} = \mathcal{J}_b^{(t)} \cup \mathcal{K}^{(t)}$
if $\mid \mathcal{J}_a^{(t)} \mid \leq \mid \mathcal{J}_b^{(t)} \mid$ **then**
    Compute: $\mathbf{M}^{(t+1)} = (\mathbf{L}^{(t+1)'}\mathbf{L}^{(t+1)})^{-1}$
else
    Compute: $\mathbf{\Lambda} = \mathbf{M}^{(t)}(\mathcal{J}_b^{(t)}, \mathcal{J}_b^{(t)})^{-1}$
    Compute: $\mathbf{\Delta} = \mathbf{M}^{(t)}(\mathcal{J}_a^{(t)}, \mathcal{J}_b^{(t)})\mathbf{\Lambda}\mathbf{M}^{(t)}(\mathcal{J}_b^{(t)}, \mathcal{J}_a^{(t)})$
    Compute: $\mathbf{M}^{(t+1)} = \mathbf{M}^{(t)}(\mathcal{J}_a^{(t)}, \mathcal{J}_a^{(t)}) - \mathbf{\Delta}$
end if
for each index k in $\mathcal{K}$ **do**
    Compute the residual: $\vec{res} = \mathbf{A}^{(t+1)}(:, k) - \mathbf{L}^{(t+1)}\mathbf{M}^{(t+1)}\mathbf{L}^{(t+1)'}\mathbf{A}^{(t+1)}(:, k)$
    if $\|\vec{res}\| \leq \varepsilon \|\mathbf{A}^{(t+1)}(:, k)\|$ **then**
        Continue;
    else
        Compute: $\delta = \|\vec{res}\|^2$; and $\vec{y} = \mathbf{M}^{(t+1)}\mathbf{L}^{(t+1)'}\mathbf{A}^{(t+1)}(:, k)$
        Update the core matrix $\mathbf{M}^{(t+1)}$: $\mathbf{M}^{(t+1)} \leftarrow \begin{pmatrix} \mathbf{M}^{(t+1)} + \vec{y}'\vec{y}/\delta & -\vec{y}/\delta \\ -\vec{y}'/\delta & 1/\delta \end{pmatrix}$
        Expand $\mathbf{L}^{(t+1)}$: $\mathbf{L}^{(t+1)} \leftarrow [\mathbf{L}^{(t+1)}, \ \mathbf{A}^{(t+1)}(:, k)]$
    end if
end for
Compute $\mathbf{R}^{(t+1)} = \mathbf{L}^{(t+1)'}\mathbf{A}^{(t+1)}$;
Update $\mathcal{J}^{(t+1)}$ and $\mathcal{K}^{(t+1)}$.

this example, it turns out that 3 is still linearly independent, whereas $l - 2$ is not any more. Additionally, some of the columns that were previously excluded as linearly dependent (e.g., 4 and 9) may now have become linearly independent, so we need to re-test those as well. In this example, it turns out that they are still redundant.

8.4.2 Proofs and Analysis

Correctness of *Colibri-D*

We have the following lemma for the correctness of Alg. 15:

Lemma 21. Correctness of *Colibri-D*. *Let the matrix $\mathbf{C}_0$ contain the initial sampled columns from $\mathbf{A}^{(t+1)}$ (i.e. $\mathbf{C}_0 = \mathbf{A}^{(t+1)}(:, \mathcal{I})$). With tolerance $\epsilon = 0$, the following facts hold for the matrices $\mathbf{L}^{(t+1)}$ and $\mathbf{M}^{(t+1)}$ in Alg. 15:*

P1: the columns of $\mathbf{L}^{(t+1)}$ *are linearly independent;*
P2: $\mathbf{L}^{(t+1)}$ *shares the same column space as* $\mathbf{C}_0$*;*
P3: the core matrix $\mathbf{M}^{(t+1)}$ *satisfies* $\mathbf{M}^{(t+1)} = (\mathbf{L}^{(t+1)'}\mathbf{L}^{(t+1)})^{-1}$.

Proof. : For 'P3', the proof is the same as the proof of 'P3' in Theorem 6.

For 'P1' we prove it by induction. First (base case), notice that $\mathbf{L}^{(t+1)}$ in step 2 is a subset of $\mathbf{L}^{(t)}$, according to Theorem 6, the columns in $\mathbf{L}^{(t+1)}$ must be linear independent with each other. Then, for the induction step (the same procedure as the proof of 'P1' in Theorem 6), we can show that every time we expand $\mathbf{L}^{(t+1)}$ (steps 15-17), the new column must be linearly independent with the current $\mathbf{L}^{(t+1)}$. Therefore, the columns in $\mathbf{L}^{(t+1)}$ is always linearly independent with each other.

For 'P2', by the proof of 'P1', we know that for each column $\mathbf{u}$ which we skip (steps 11-13), it can be expressed as a linear combination of the current columns in $\mathbf{L}^{(t+1)}$. In other words, $\mathbf{u}$ is linearly redundant with respect to $\mathbf{L}^{(t+1)}$. Therefore, $\mathbf{C}_0$ and $\mathbf{L}^{(t+1)}$ share the same column space, which completes the proof. □

By Lemma 21 and Theorem 6, the three matrices $\mathbf{L}^{(t+1)}$, $\mathbf{M}^{(t+1)}$, and $\mathbf{R}^{(t+1)}$ produced by Alg. 15 are exactly the same as if we had called Alg. 14 for time step $(t+1)$ from the scratch. Therefore, we have the following corollary:

Corollary 7. *Using the same index set* $\mathcal{I}$ *of initial sampled columns for all time steps, Alg. 15 has exactly the same approximation accuracy as Alg. 14, CUR [DKM05b] and CMD [SXZF07].*

Efficiency of *Colibri-D*

Since the three matrices $\mathbf{L}^{(t+1)}$, $\mathbf{M}^{(t+1)}$, and $\mathbf{R}^{(t+1)}$ by Alg. 15 are exactly the same as if we had called Alg. 14 for time step $(t+1)$, we have the following corollary for the space cost of Alg. 15:

Corollary 8. *Using a fixed indices set* $\mathcal{I}$ *of initial sampled columns, the space cost of Alg. 15 is the same as Alg. 14 and it is equal or better compared to CUR [DKM05b] and CMD [SXZF07].*

We have the following lemma about the speed of Alg. 15.

Lemma 22. Efficiency of *Colibri-D*. *Let* $r_1 = |\ \mathcal{J}_a^{(t)}\ |$, $r_2 = |\ \mathcal{J}_b^{(t)}\ |$ *and* $r_3 = |\ \mathcal{K}^{(t)}\ |$. *The computational complexity of Alg. 15 is bounded by* $O(\max(r_1, r_2, r_3)^3 + (r_2 + r_3)\tilde{m}^{(t+1)})$, *where* $\tilde{m}^{(t+1)}$ *is number of edges in the matrix* $\mathbf{L}^{(t+1)}$.

Proof. : The cost of steps 1-2 is constant. The cost of steps 3-9 is $O(\max(r_1, r_2, r_3)^3)$. For the cost of steps 10-19, we can show that (same as the proof of Lemma 17), it is $O((r_2+r_3)\tilde{m}^{(t+1)})$. Putting them together, we have that the total cost of Alg. 15 is $O(\max(r_1, r_2, r_3)^3 + (r_2+r_3)\tilde{m}^{(t+1)})$, which completes the proof. □

In terms of speed, the difference between Alg. 15 and Alg. 14 lies in the different way of initializing the matrix $\mathbf{M}^{(t+1)}$ (steps 3–9 of Alg. 15). More specifically, if $r_1 \leq r_2$, the computational cost for initializing $\mathbf{M}^{(t+1)}$ is asymptotically the same for both algorithms—both are $O(r_1^3)$. On the other hand, if $r_1 > r_2$, we only need $O(r_1^2 r_2)$ for Alg. 15 while Alg. 14 still requires $O(r_1^3)$. Based on this fact as well as Lemma 17, we have the following corollary.

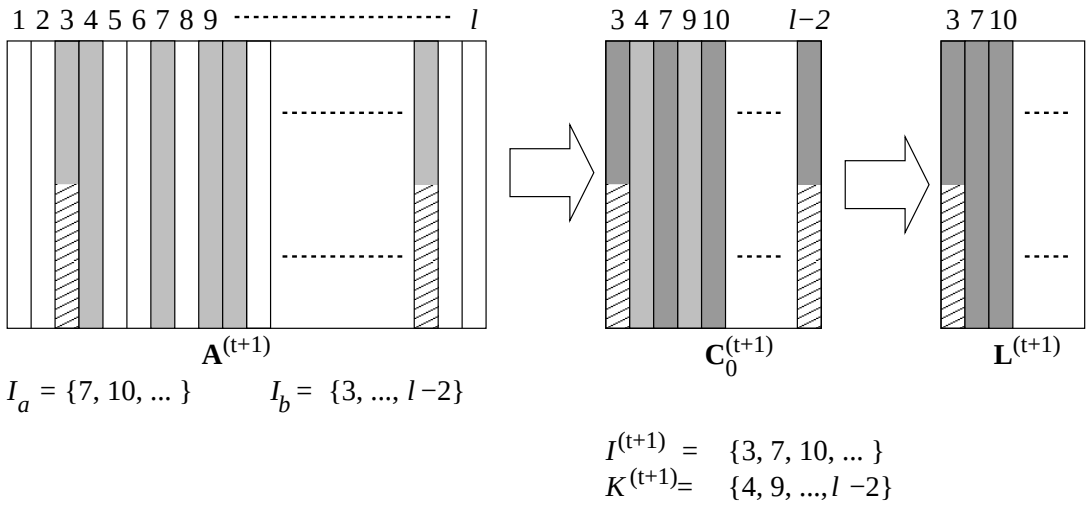


Figure 8.4: Illustration of notation and process for *Colibri-D*—compare with Figure. 8.3. Shaded and dark shaded columns as in Figure. 8.3, shaded and filled columns are those from the previous timestep that contain at least one new entry.

Corollary 9. *Using a fixed set $\mathcal{I}$ of initial sampled columns, the running time of Alg. 15 is equal or better compared to Alg. 14, CUR [DKM05b] and CMD [SXZF07].*

To summarize, if we fix the index set $\mathcal{I}$ of initial sampled columns for all time steps, the proposed Alg. 15 will produce the low rank approximation at each time step t with the same accuracy as CUR/CMD and our own Alg. 14 for static graphs. For both speed and space cost, it is always equal or better than CUR/CMD as well as our Alg. 14.

8.5 Applications: Case Studies

As mentioned before, low rank approximations constitute a powerful tool to mining both static and dynamic graphs. Notice that our algorithms can achieve the same approximation accuracy as CUR/CMD. Thus, in principle, we can do whatever CUR/CMD can do, only that our methods will probably be much faster and nimbler. Next, we present two examples as case studies: community tracking (subsection 8.5.1) and anomaly detection (subsection 8.5.2).

8.5.1 Community Tracking

The low rank approximation of the adjacency matrix $\mathbf{A}^{(t)}$ often reveals the community structure in the graphs. Therefore, by tracking the low rank approximation of $\mathbf{A}^{(t)}$ over time ($t = 1, 2, \ldots$), we can monitor the community structure (see Figure 8.5).

There are numerous graph partitioning and community detection algorithms. We believe that several of them would benefit from a good, low-rank approximation. To illustrate the ability of *Colibri* to find and monitor communities, among the many choices, we use Algorithm 16. Here we

apply *Colibri-D* to a sequence of the adjacency matrices $\mathbf{A}^{(t)}$ for each time step $t \geq 1$. Note that, in contrast to CUR/CMD, the columns in $\mathbf{L}^{(t)}$ are linearly independent, which serve as basis vectors. The projections onto the basis vectors give us the low-dimensional feature vectors for each nodes, aka the columns of $\mathbf{ML}$. We then perform k-means on them to generate the clustering result.

Algorithm 16 Community Tracking over Time

Require: The adjacency matrix $\mathbf{A}^{(t)}(t = 1, 2, ...)$, sample size c and ε
Ensure: The community at each time t for the given graph.

```
for t = 1, 2, ... do
    if t == 1 then
        set the low rank approximation for A^1 by Alg. 14. Let the output of Alg. 14 be L^(t), M^(t) and R^(t);
    else
        Update low rank approximation for A^t by Alg. 15. Let the output of Alg. 15 be L^(t), M^(t) and R^(t);
    end if
    Let X = M^(t) R^(t); and k be the number of columns in L^(t);
    Treat each column in X as a feature vector for the corresponding node.
    Use k-means to cluster X into k clusters.
end for
```

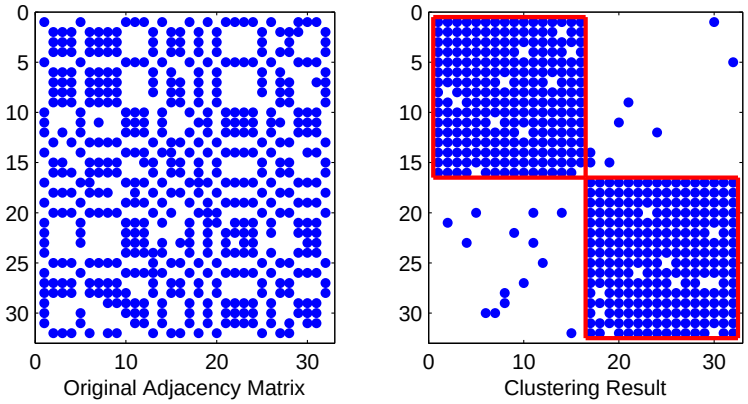
Note that Alg. 16 assumes that the number of communities equals the number of columns in the matrix $\mathbf{L}^{(t)}$. In practice, given the sample size c, we can control the number of communities by choosing different $\varepsilon \neq 0$. In this way, we also want to eliminate those 'nearly linearly dependent' columns in the matrix $\mathbf{L}^{(t)}$.

Figure 8.5 gives an example of applying Alg. 16 to a synthetic dataset, a sequence of so-called "Cavemen" graphs. These graphs are almost block-diagonal, and their name comes from social networks, where a group of hypothetical cavemen tend to know almost everybody else in their cave, but few cavemen from the other cave(s). We present the results for three time steps. Each sub-figure in the left column is the original adjacency matrix at that time step, and the sub-figure in the right column is the clustering result (the adjacency matrix after re-ordering the nodes belonging to the same clustering together). We set the sample size $c = 15$ and $\varepsilon = 0.5$ [6]. Alg. 16 naturally tracks the evolution of the communities over time: it starts with two large communities; then a third one emerges, and then the middle community is absorbed in the first one.

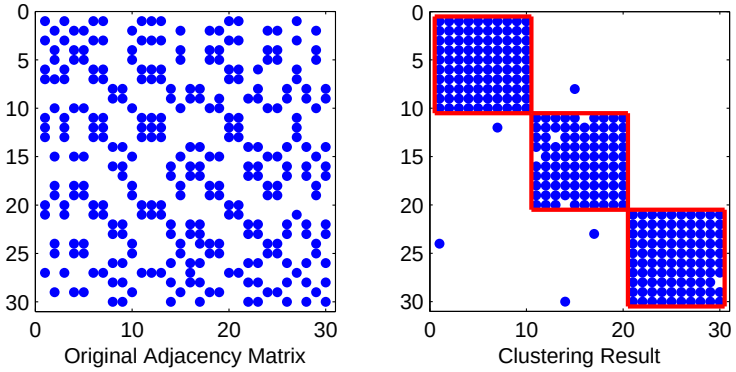
8.5.2 Anomaly Detection

We can also use *Colibri* to detect anomalies in the graphs. In [SXZF07], the authors discussed various ways to detect anomaly network traffic by CMD. The basic idea is to examine the reconstruction error. For example, a large reconstruction error for a specific row often indicates abnormal source hosts (e.g. port scanners who send traffic to many different hosts). Similarly,

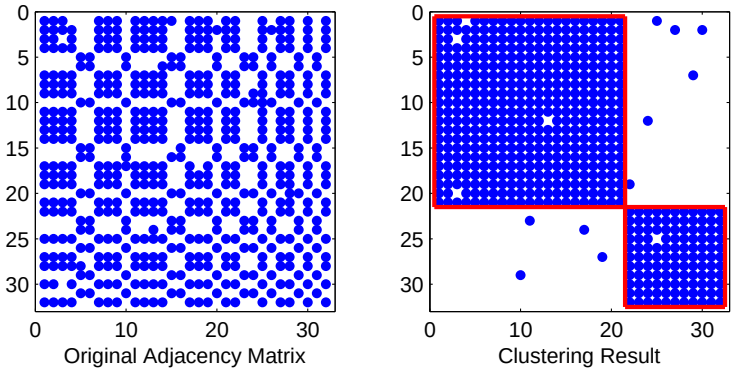
[6] How to choose an optimal ε in the general case is outside the scope of this paper.



(a) $t = 1$



(b) $t = 2$.



(c) $t = 3$.

Figure 8.5: An example of applying Alg. 16 to a dynamic Caveman graph. The sub-figures in the left column are the original adjacency matrices $\mathbf{M}^{(t)}$ with permutation. The sub-figures in the right column are the adjacency matrices after reordering by our *Colibri*. With the sample size $c = 15$, and $\varepsilon = 0.5$, we can track the evolution of the communities over time.

a large reconstruction error for a specific column often implies abnormal destination hosts (e.g. targets of distributed denial of service attacks (DDoS)). Furthermore, a large reconstruction error for the whole adjacency matrix might indicate some global anomaly (e.g. the onset of worm-like hierarchical scanning activities).

Since *Colibri* shares the exactly same reconstruction error as CMD, it is able to detect *all* these abnormal behaviors as CMD does. Figure 8.6 presents such an example. We plot the reconstruction accuracy for a given column (i.e. destination host) for the Network Traffic data over 20 hours. We manually inject the anomalies into the given column in the 13^{th} hour (marked by the dashed circle), exactly as in [SXZF07]. From Figure 8.6 we see a clear drop of the reconstruction accuracy for the given destination host, exactly at the time we injected the anomaly (the 13^{th} hour). Note that while both *Colibri* and CMD will output exactly the same curve if we use the same initial sampled columns, *Colibri* is often significantly faster, as we show next.

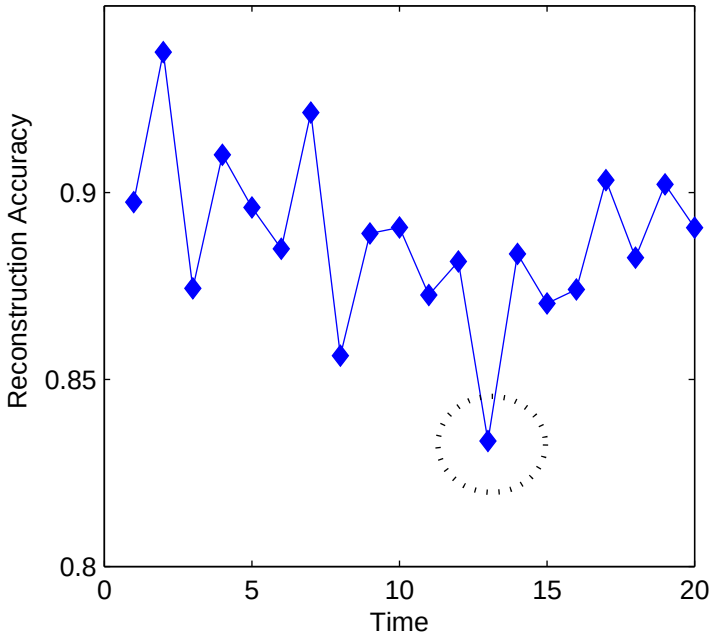


Figure 8.6: An example of applying *Colibri* to detect abnormal destination hosts. A big drop in the reconstruction accuracy (marked by dash circle), which is found by our *Colibri*, corresponds to the exact time step when we inject the anomalies.

8.6 Experimental Evaluations

Here we give experimental results for the proposed *Colibri*. Our evaluation mainly focuses on (1) the reconstruction accuracy, (2) the running time and (3) the space cost. After a brief introduction

of the datasets and the evaluation criteria, we give the results for *Colibri-S* in subsection 8.6.2, and for *Colibri-D* in subsection 8.6.3.

8.6.1 Experimental Setup

We use a network traffic dataset from the backbone router of a class-B university network. We create a traffic matrix for every hour, with the rows and columns corresponding to the IP sources and IP destinations. We turn the matrix into a binary matrix, that is, a '1' entry means that there is some TCP flow from the corresponding IP source to the destination within that hour. In short, we ignore the volume of such traffic. Overall there are 21,837 different source/destination pairs, 1,222 consecutive hours and 22.8K edges per hour, on average.

Let $\tilde{\mathbf{A}} = \mathbf{LMR}$. We use the standard reconstruction accuracy to measure the approximation quality (exactly as in [SXZF07]), to estimate the SSE, the sum-squared-error, with sample size c=1,000 for both rows and columns:

$$\begin{aligned} \text{Accu} &= 1 - \text{SSE} \\ &= 1 - \sum_{i,j} (\mathbf{A}(i,j) - \tilde{\mathbf{A}}(i,j))^2 / (\sum_{i,j} \mathbf{A}(i,j)^2) \end{aligned} \tag{8.6}$$

For a given low rank approximation $\{\mathbf{L}_{n\times\tilde{c}},\ \mathbf{M}_{\tilde{c}\times\tilde{c}},\ \mathbf{R}_{\tilde{c}\times l}\}$, the matrices $\mathbf{L}$ and $\mathbf{R}$ are usually sparse, and thus we store them as adjacency lists. In contrast, the matrix $\mathbf{M}$ is usually dense, and we store it as a full matrix. Thus, the space cost is:

$$\text{SPCost} = \text{NNZ}(\mathbf{L}) + \text{NNZ}(\mathbf{R}) + \tilde{c}^2 \tag{8.7}$$

where $\text{NNZ}(.)$ is the number of non-zero entries in the matrix.

For the computational cost, we report the wall-clock time. All the experiments ran on the same machine with four 2.4GHz AMD CPUs and 48GB memory, running Linux (2.6 kernel). For each experiment, we run it 10 times and report the average.

Notice that for both Theorem 1 and Lemma 5, we require the tolerance $\varepsilon = 0$. In our experiments, we find by changing ε to be a small positive number (e.g., $\varepsilon = 10^{-6}$), it does not influence the approximation accuracy (up to 4 digits precision), while it makes the proposed algorithms more numerically stable[7]. Therefore, for all the experiments we reported in this paper, we use $\varepsilon = 10^{-6}$ for both *Colibri-S* and *Colibri-D*.

8.6.2 Performance of *Colibri-S*

Here, we evaluate the performance of our *Colibri-S* for static graphs, in terms of speed and space.

We compare *Colibri-S* against the best published techniques, and specifically against CUR [DKM05b] and CMD [SXZF07]. For brevity and clarity, we omit the comparison against SVD, because

[7]this is an implementation detail. We omit the detailed discussion due to the space limit. How to choose an optimal ε is out-of the scope of this paper.

CMD [SXZF07] was reported to be significantly faster and nimbler than SVD, with savings up to 100 times.

We aggregate the traffic matrices within the first 100 hours and then ignore the edge weights as the target matrix $\mathbf{A}$. Totally, there are 158,805 edges in this graph. We vary the sample size c from 1,000 to 8,000, and study how the accuracy changes with the running time and space cost for all three methods.

Figure 8.7 plots the mean running time vs. the approximation accuracy. Notice that the y-axes is in the logarithm scale. *Colibri-S* is significantly faster than both CUR and CMD, by 28x$\sim$353x and 12x$\sim$52x respectively.

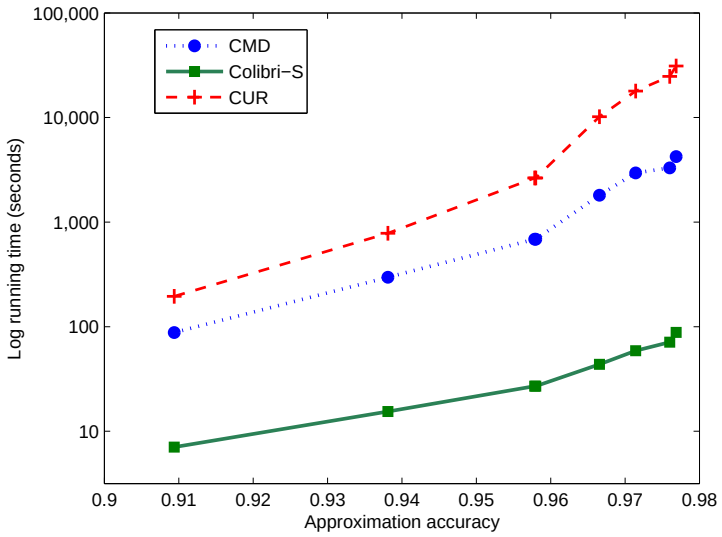


Figure 8.7: Running time vs. accuracy. Our *Colibri-S* (in green squares) is significantly faster than both CUR and CMD, for the same approximation accuracy. Note that the y-axis is in logarithmic scale.

With respect to space cost, CUR is always the most expensive among the three methods and therefore we use it as the baseline. Figure 8.8 plots the relative space cost of CMD and *Colibri-S*, vs. the approximation accuracy. Again, *Colibri-S* outperforms both CUR and CMD. Overall, *Colibri-S* only requires 7.4%$\sim$28.6% space cost of CUR, and 28.6%$\sim$59.1% space cost of CMD for the same approximation accuracy.

The reader may be wondering what causes all these savings. The answer is the reduction in columns kept: in *Colibri-S* we only keep those linearly independent columns, and discard all the other of the c columns that CUR chooses (and keeps). This idea eventually leads to significant savings. For example, with a sample size of $c = 8,000$ (the number of columns that CUR will keep), CMD discards duplicates, keeping on the average only $3,220$ unique columns, and *Colibri-S* further discards the linearly dependent ones, eventually keeping only $1,101$. And, thanks to our Theorem 6, the columns that *Colibri-S* discards have no effect on the desired subspace, and neither on the approximation quality.

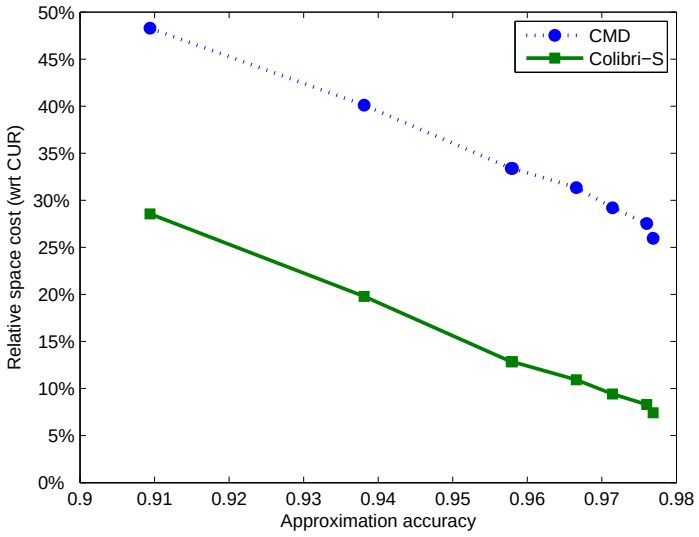


Figure 8.8: Relative space cost of *Colibri-S* and CMD, versus accuracy. Space costs are normalized by the space of CUR. *Colibri-S* consistently requires a fraction of the space by CUR/CMD, for same accuracy.

8.6.3 Performance of *Colibri-D*

We use the same aggregated traffic matrix as in subsection 8.6.2; and initialize the algorithm by a sample size $c = 2,000$ (which gives an average accuacy of 93.8%). Then, we randomly perturb r out of these 2,000 sampled columns and update the low rank approximation of the updated adjacency matrix. Since *Colibri-D* has the same space cost as *Colibri-S*, we only present the results on the running time.

We compare our *Colibri-D* against both CMD and against our own *Colibri-S* We apply CMD and *Colibri-S* for each (static) instance of the graph and report the wall-clock times. For visual clarity, we omit the comparison against CUR, since it is consistently slower than both CMD and *Colibri-S* on static graphs, as shown in subsection 8.6.2.

Figure 8.9 plots the wall-clock time of CMD, *Colibri-S* and *Colibri-D*, versus r (the number of updated columns). *Colibri-D* is 2.5x$\sim$112x faster than CMD. Even compared against our own *Colibri-S* *Colibri-D* is still about 2x$\sim$5x faster. The computational savings of *Colibri-D* over *Colibri-S* come from the Sherman-Morrison Lemma: if the graph evolves smoothly, *Colibri-D* leverages the low rank approximation of the previous time step, and does a fast (but exact) update. We repeat that all three methods have *identical* approximation accuracy, if they use the same initial sampled columns.

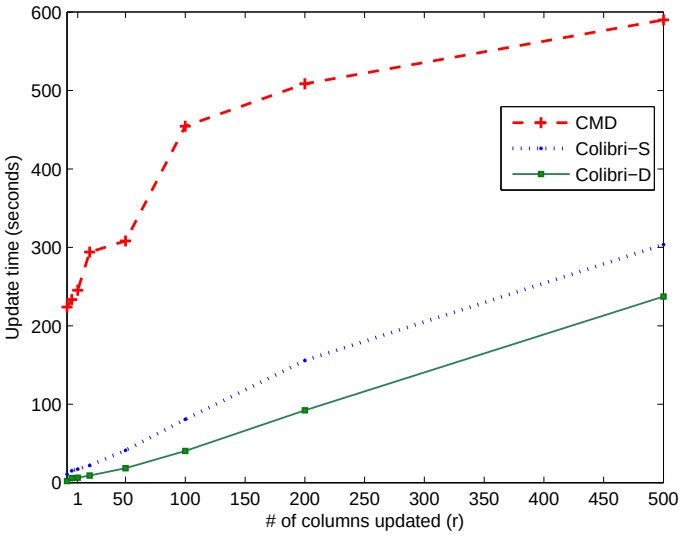


Figure 8.9: Performance for dynamic graphs: Speed versus number of updated columns. *Colibri-D* (in green squares) is 2.5x∼112x faster than the best published competitor (CMD); and also faster than our own *Colibri-S*, applied on each individual graph instance.

8.7 Related Work

In this section, we briefly review the related work on matrix low rank approximation. For the related work on general graph mining, please refer to Chapter 6.

For static graphs, the most popular choices include SVD/PCA [GVL89, KAS98] and random projection [Ind00]. However, these methods often ignore the sparseness of many real graphs and therefore often need huge amount of space and processing time (See [SXZF07] for a detailed evaluation). More recently, Drineas et al [DKM05b] proposed the CUR decomposition, which partially deals with the sparsity of the graphs. CUR is proved to achieve an optimal approximation while maintain the sparsity of the matrix. Sun et al [SXZF07] further improve CUR by removing the duplicate columns/row in the sampling stage. Their method, named as CMD, is shown to produce the same approximation accuracy, but it often requires much less time and space. Our method (*Colibri-S*) further improves the efficiency in speed and space by leveraging the linear correlation among different sampled columns. As a result, our method saves the computational time and space cost, while it outputs exactly the same low rank approximation as CUR/CMD.

The worst-case computational complexity of CUR, CMD and *Colibri* is linear to the size of the matrix. A more accurate CUR approximation has been proposed in [DMM07], but it requires SVD operation on the whole matrix as a preprocessing step which is often too expensive for many large scale applications.

For dynamic graphs, a lot of SVD based techniques have been proposed, such as multiple time series mining [GGK03, PSF05], dynamic tensor analysis [STF06], incremental spectral clustering [NXC+07] etc. As for the static graphs, these methods might suffer from the loss-of-sparsity

issue for large sparse graphs despite their success in the general cases. Sun et al [SXZF07] deal with this issue by applying their CMD method independently for each time step. However, how to make use of the smoothness between two consecutive time steps to do even more efficient computation is not exploited in [SXZF07]. This is exactly the unique feature of our *Colibri-D*, - it leverages such smoothness to do fast update while maintaining the sparseness of the resulting low rank approximation.

8.8 Conclusion

In this chapter, we propose the family of *Colibri* methods to do fast mining on large static and dynamic graphs. The main contributions of the paper are:

- A family of novel, low rank approximation methods (*Colibri-S*, *Colibri-D*) for static and dynamic graphs, respectively: *Colibri-S* saves space and time by eliminating linearly dependent columns; *Colibri-D* builds on *Colibri-S*, and performs incremental updates efficiently, by exploiting the "smoothness" between two consecutive time steps.
- Proofs and complexity analysis, showing our methods are provably equal or better compared to the best known methods in the literature, while maintaining exactly the same accuracy;
- Extensive experimental evaluation, showing that our methods are significantly faster and nimbler than the state of the art (up to *112* times faster). See Figure 8.1 for comparisons against CUR [DKM05b] and CMD [SXZF07].

Chapter 9

Mining Complex Time-Stamped Events

Summary of This Chapter

- **Questions we want to answer:**
  - Q: How to mine complex time-stamped events (e.g., find similar time stamps, abnormal time stamps as well as the interpretations for our findings)?
- **Our answers and contributions**
  - A1: We proposed a generic framework (T3) to mine complex time-stamped events.
  - A2: We developed an efficient algorithm (MT3) for multiple scale analysis.

9.1 Introduction

In many real applications, data sets are often collected at different time stamps. At each time stamp, we might observe a set of events, where each event consists of a set of entities. Furthermore, each entity can have its own attributes. For example, in social networks, we might observe activities (events) at each day (time), where each activity involves a set of different people (entities) – each with his/her own attributes (e.g., job title). Another example is the yearly DBLP data sets, where a time stamp is 'publish year'; an event is a 'paper'; and entities are 'author,' 'conference,' etc.

How can we analyze time in such a complex context. For example, are there any two time stamps that look similar with each other? Can we find any abnormal time stamp whose behavior is very different from other time stamps? How can we interpret our findings? Furthermore, how can we do such analysis on multiple scales in an efficient way?

In this chapter, we address the above challenges in multiple dimensions. First in a single scale, our method (T3) can automatically group time stamps into meaningful clusters as well as spot the abnormal stamps. For each cluster/abnormal time stamp, it also outputs the selective subsets of events/entities/attribute values as their interpretations. Here, the main idea is (1) to adopt a graph representation for the data sets at different time stamps and (2) to explore the proximity among different nodes (time/events/entities/ attribute values), based on this we will find clusters

and anomalies as well as their interpretations. Our experiments on several real data sets demonstrate that T3 always outputs results (i.e., clusters and anomalies as well as their interpretations) that are consistent with human intuitions. Furthermore, we propose MT3 to allow efficient analysis on multiple scales. Here, the key idea is to explore the "smoothness" (i.e., redundancy) among different scales. Our experiments show that MT3 leads to exactly the same results (i.e, *no quality loss*), but achieves significant speed-ups (up to *2 orders of magnitude*).

The main contributions of this chapter are summarized as follows:

- A generic framework (T3) to mine complex time-stamped events in complex context
- An efficient algorithm (MT3) for multiple scale analysis
- Power of our approach illustrated by extensive experiments on several real datasets

The rest of this chapter is organized as follows. We begin in Section 2 with the formal problem definition. We present T3 for the single scale analysis and MT3 for the multiple scale analysis in Section 3 and Section 4, respectively. The experimental results are reported in Section 5. We review the related work in Section 6 and conclude the chapter in Section 7.

9.2 Problem Definition

In this section, we first introduce our notations and data representation, and then give the formal problem definitions.

Table 9.2 lists the main symbols we use throughout this chapter. Following standard notation, we use calligraphic letter for sets (e.g., $\mathcal{O}^1$ is the set of all time stamps), capital bolded letters for matrices (e.g., $\mathbf{W}$), and lower case bolded letters for vectors (e.g., $\mathbf{g}$). We denote the transpose with a prime (i.e., $\mathbf{W}'$ is the transpose of $\mathbf{W}$), and we use superscripts to denote the indices for object types (e.g., $\mathcal{O}^s$ is the s^{th} type of object) and the indices for block matrices (e.g., $\mathbf{W}^{x,y}$ is a block matrix of the matrix $\mathbf{W}$). For matrix/vector, we use the subscript to represent the size of the matrix/vector (e.g. $\mathbf{0}_{k\times l}$ means a matrix of size $k \times l$, whose elements are all zero). If the size of a matrix/vector is clear from the context, we omit such subscripts. Also, we represent the elements in a matrix using a convention similar to Matlab, e.g., $\mathbf{W}(i,j)$ is the element at the i^{th} row and j^{th} column of the matrix $\mathbf{W}$, and $\mathbf{W}(i,:)$ is the i^{th} row of $\mathbf{W}$, etc.

In our setting, the datasets are collected at different time stamps. At each time stamp, we observe a set of events, where each event consists of a set of entities. Furthermore, each entity may or may not have its own attributes. For example, in the running example in Table 9.2(a), we observe 9 events ($e_1, ..., e_9$), each of which is a social event (e.g., e_1 is a 'technical meeting', e_2 is a 'football game', etc). The events are spreaded among 6 time stamps ($t_1, ..., t_6$), each of which is a day (e.g., t_1 is 'Monday', t_2 is 'Tuesday', etc). Furthermore, each event involves 2 entities ($b_1, ..., b_8$), each of which is a person (e.g., b_1 is 'John', b_2 is 'Smith', etc) .

To simplify the description, we refer to 'time', 'event', each type of 'entity', and each 'attribute' as one type of object, respectively. If we have p types of entities (in the running example, $p = 1$), and q types of attributes (in the running example, $q = 0$), we define the following object set $\mathcal{O}^x(x = 1, ..., 2 + p + q)$, where the first type of object is always 'time'; the second type of object is always 'event'; each of the next p objects is one type of 'entity'; and each of the next q objects

Table 9.1: Symbols

Symbol	Definition and Description
$\mathcal{O}^1$	the 'time' object: $\mathcal{O}^1 = \{t_1, ..., t_{n_1}\}$
$\mathcal{O}^2$	the 'event' object: $\mathcal{O}^2 = \{e_1, ..., e_{n_2}\}$
$\mathcal{O}^x$	the $(x-2)^{\text{th}}$ 'entity' object: $\mathcal{O}^x = \{b_1^{(x-2)}, ..., b_{n_x}^{(x-2)}\}, (x = 3, ..., 2+p)$
$\mathcal{O}^y$	the $(y-2-p)^{\text{th}}$ 'attribute' object: $\mathcal{O}^y = \{a_1^{(y-2-p)}, ..., a_{n_y}^{(y-2-p)}\}$, $(y = 3+p, ..., 2+p+q)$
$\mathbf{W}^{x,y}$	the adjacency matrix ($n_x \times n_y$) from the x^{th} object to the y^{th} object $(x, y = 1, ..., 2+p+q)$
$\mathbf{D}^{x,y}$	the degree matrix: $\mathbf{D}^{x,y}(i,i) = \sum_j \mathbf{W}^{x,y}(i,j)$ and $\mathbf{D}^{x,y}(i,j) = 0 (i \neq j)$
$\mathbf{W} = [\mathbf{W}^{x,y}]$	the overall adjacency matrix ($n \times n$)
$\mathbf{0}$	a matrix with all elements equal to 0
$\mathbf{I}$	an identity matrix
p	the number of different types of entities
q	the number of different types of attributes
n_x	the number of instances for the x^{th} type of object $(x = 1, ..., 2+p+q)$
n	the number of total instances ($n = \sum_{x=1}^{2+p+q} n_x$)
s_x	the number of objects connected to the x^{th} type of object
z	the number of clusters for time stamps
$r_{i,j}$	the proximity score from node j to node i
c	$(1-c)$ is the restart probability for random walk with restart ($c = 0.95$ in this chapter.)
$\mathbf{ttP} = [r_{i,j}]$	the time-to-time proximity matrix ($n_1 \times n_1$, and $i, j = 1, ..., n_1$)
$\mathbf{toP} = [r_{i,j}]$	the time-to-others proximity matrix ($(n-n_1) \times n_1$, and $i = 1, ..., n-n_1, j = 1, ..., n_1$)
$\mathbf{f}$	the aggregation function ($n_1 \times 1$ vector)
$\mathbf{g}$	the cluster membership function ($n_1 \times 1$ vector)

is one type of 'attribute'. For the running example in Table 9.2(a), we have 3 types of objects in the object set $\mathcal{O}^x (x = 1, 2, 3)$. They are 'time', 'event', and 'entity', respectively. (There is no 'attribute' in this example.) Each object type has a set of instances. For example, the instances for the 'time' object ($\mathcal{O}^1$) are different time stamps (e.g., $t_1, t_2, ...$).

In this chapter, we use a graph representation for the whole dataset covering all time stamps. To be specific, we treat each instance for each type of object as a node in the graph. For example, Table 9.2(b) gives the graph representation for the original time-stamped datasets (depicted in Table 9.2(a)) – where each time stamp, each event instance, and each entity instance is represented as a single node in the graph. Furthermore, the relationship between different types of objects are modeled by the adjacency matrices ($\mathbf{W}^{x,y}(x, y = 1, ..., 2+p+q)$). For example, we can use $\mathbf{W}^{1,2}$ to model the relationship between the 'time' object and 'event' object, where $\mathbf{W}^{1,2}(i,j) = 1$ iff the j^{th} event happens at the i^{th} time stamp; $\mathbf{W}^{1,2}(i,j) = 0$ otherwise. Similarly, we can use $\mathbf{W}^{2,2+x}(x = 1, .., p)$ to model the relationship between the 'event' object and the x^{th}

'entity' object, where $\mathbf{W}^{2,2+x}(i,j) = 1$ iff the i^{th} event involves the j^{th} instance of the x^{th} type of entity; $\mathbf{W}^{2,2+x}(i,j) = 0$ otherwise. We can use $\mathbf{W}^{2+x,2+p+y}(x = 1, ..., p, y = 1, ..., q)$ to model the relationship between the x^{th} type of 'entity' object the y^{th} type of 'attribute' object, where $\mathbf{W}^{2+x,2+p+y}(i,j) = 1$ iff the i^{th} instance of the x^{th} type of 'entity' has the j^{th} attribute value of the y^{th} type of 'attribute'; $\mathbf{W}^{2+x,2+p+y}(i,j) = 0$ otherwise. For the running example, two such adjacency matrices ($\mathbf{W}^{1,2}$ and $\mathbf{W}^{2,3}$) are enough to model all the relationships (see Table 9.2(c)).

If we always reserve the first n_1 rows/columns for the time nodes; the next n_2 rows/columns for the event nodes; followed by rows/ columns for entity nodes and attribute nodes respectively; we can define $\mathbf{W} = \mathbf{W}^{x,y}$ $(x, y = 1, ..., 2+p+q)$ as the overall adjacency matrix for the whole graph. Note that if there is no relationship between the x^{th} and the y^{th} objects, the corresponding block matrix $\mathbf{W}^{x,y} = \mathbf{0}$. Also, by this notation, we allow additional relationship within the same type of object. For example, if we want to consider the continuous property of time, we can put extra links between consecutive time nodes, which will lead to a non-zero block matrix $\mathbf{W}^{1,1}$. For the running example in Table 9.2, its overall adjacency matrix $\mathbf{W}$ has the following format (Eq. (9.1)):

$$\mathbf{W} = \begin{pmatrix} \mathbf{0} & \mathbf{W}^{1,2} & \mathbf{0} \\ (\mathbf{W}^{1,2})' & \mathbf{0} & \mathbf{W}^{2,3} \\ \mathbf{0} & (\mathbf{W}^{2,3})' & \mathbf{0} \end{pmatrix} \tag{9.1}$$

With the above notation, our datasets can be denoted by the object set $\mathcal{O}^x(x = 1, ..., 2+p+q)$ together with the overall adjacency matrix $\mathbf{W}$. Our goal is to find (1) similar/anomalous time stamps and (2) their interpretations. In this chapter, we define an anomalous time stamp as a special time cluster, which contains a single time stamp. Therefore, we define the cluster membership function $\mathbf{g}$ as an $n_1 \times 1$ vector, and each element in $\mathbf{g}$ as an integer between 1 and z (z is the cluster number for time stamps), indicating to which cluster it belongs. To provide an interpretation for each time cluster, we want to select a representative subset of instances from each type of object (except 'time' object). Thus, our problem (*The Single Scale Analysis*) can be formally defined as follows:

Problem 12. The Single Scale Analysis

Given: *The datasets collected at different time stamps:* $\{\mathcal{O}^x, \mathbf{W}\}(x = 1, .., 2+p+q)$.

Find: *(i) The cluster membership function* $\mathbf{g}$ *for time stamps (as well as the cluster number* z*); and (ii) for each time cluster, a representative subset of instances from each type of object (except 'time' object).*

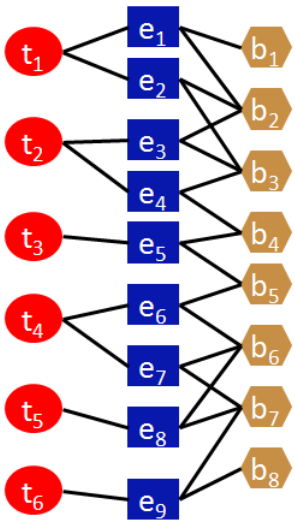
For example, Figure 9.1(a) shows the output of the proposed T3 (for the single scale analysis) applied to the datasets we list in Table 9.2, where we find 2 clusters of time stamps ($\{t_1, t_2\}$ and $\{t_4, t_5, t_6\}$) and 1 abnormal time stamp (t_3). Therefore, our cluster membership function satisfies: $\mathbf{g} = [1,\ 1,\ 3,\ 2,\ 2,\ 2]'$. For each time cluster as well as the abnormal time stamp, we also output a representative subset of the entity nodes as its interpretations.[1]

Besides the finest scale, we might also want to do the same analysis (i.e., to find the time cluster/anomaly as well as their interpretations) on some coarser scale. To this end, we introduce

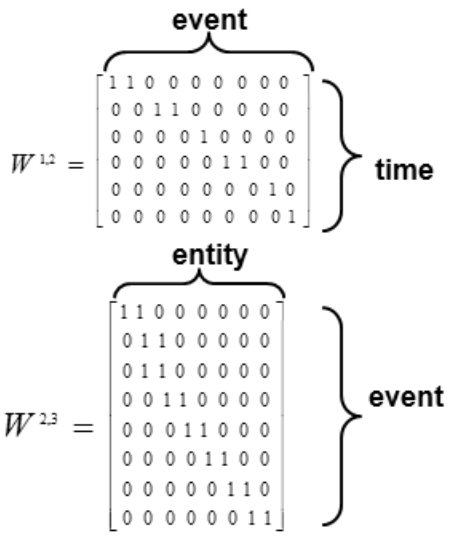
[1]For the sake of simplicity, the representative events are not shown in the figure.

Time Steps	Events	Entities
t_1	e_1	b_1, b_2
	e_2	b_2, b_3
t_2	e_3	b_2, b_3
	e_4	b_3, b_4
t_3	e_5	b_4, b_5
t_4	e_6	b_5, b_6
	e_7	b_6, b_7
t_5	e_8	b_6, b_7
t_6	e_9	b_7, b_8

(a) Original data sets



(b) Graph representation



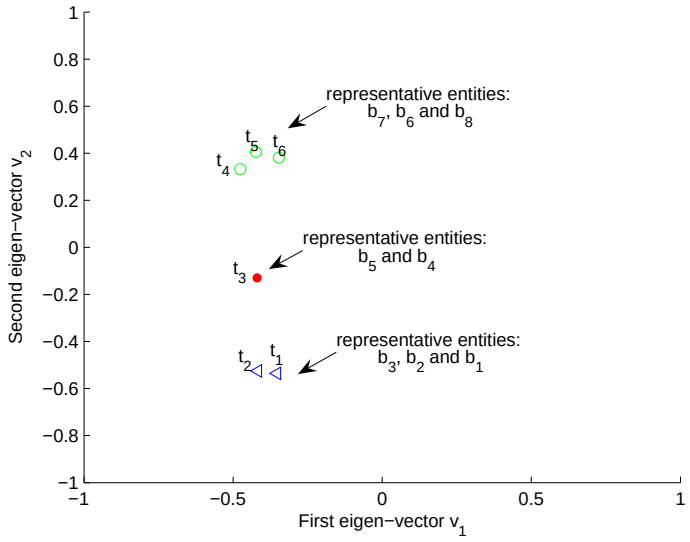
(c) Adjacency matrices

Table 9.2: A running example: notations and representation illustration.

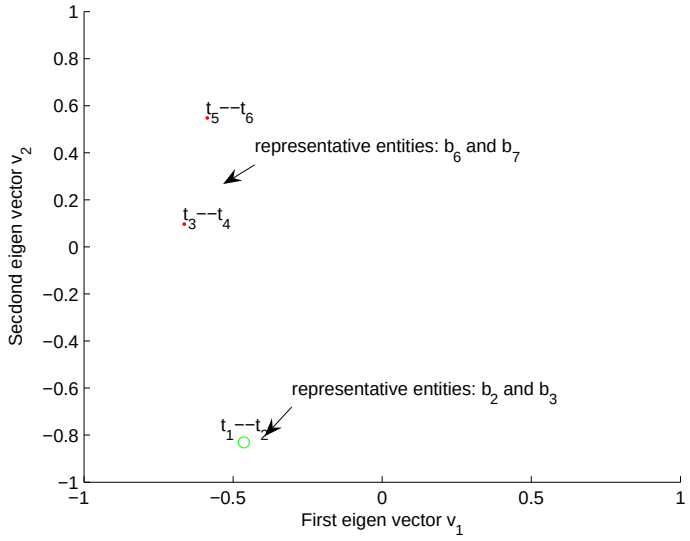
the aggregation function $\mathbf{f}$, which is an $n_1 \times 1$ vector. For example, if we aggregate the time by every two time stamps for the datasets in Table 9.2, the aggregation function $\vec{u}$ is a 6×1 vector: $\mathbf{f} = [\mathbf{1}, \mathbf{1}, \mathbf{2}, \mathbf{2}, \mathbf{3}, \mathbf{3}]'$. Also, let $\tilde{\mathbf{g}}$ be the cluster membership function and $\tilde{z}$ be the cluster number at the aggregated scale, respectively. With this notation, our problem (*The Multiple Scale Analysis*) can be formally defined as follows:

Problem 13. The Multiple Scale Analysis

Given: *(i) The datasets collected at different time stamps:*



(a) The finest scale



(b) The aggregated scale (by every two time stamps)

Figure 9.1: The outputs for the running example in Table 9.2.

$\{\mathcal{O}^x, \mathbf{W}\}(x = 1, .., 2 + p + q)$*; and (ii) the aggregation function* $\mathbf{f}$.

Find: *(i) The cluster membership function* $\tilde{\mathbf{g}}$ *for time stamps (as well as the cluster number* $\tilde{z}$*); and (ii) for each time cluster at aggregated scale, a representative subset of instances from each type of object (except 'time' object).*

For example, Figure 9.1(b) shows the output of the proposed MT3 applied to the datasets in Table 9.2 if we aggregate the time by every two time stamps. Notice that in this case, the abnormal time stamp (i.e., t_3 at the finest scale) disappears.

9.3 T3 for Single Scale Analysis

In this section, we propose T3 to address Problem 12. We first give an overview of the proposed algorithm (T3), and then introduce each component of T3 in detail.

9.3.1 Overview of T3

Alg. 17 gives the overview of the proposed T3 for single scale analysis. In T3, we first construct the graph representation $\mathbf{W}$ from the original raw datasets as introduced in Section 2 (step 1). Then (step 2), we will compute two proximity matrices from the adjacency matrix $\mathbf{W}$: the time-to-time proximity matrix ($\mathbf{ttP}$) and the time-to-others proximity matrix ($\mathbf{toP}$). The time-to-time proximity matrix ($\mathbf{ttP}$) will be used to find the time cluster membership function $\mathbf{g}$ (step 3); while the time-to-others proximity matrix ($\mathbf{toP}$) will be used to find the representative subset of instances as the interpretations for time cluster (step 4).

Algorithm 17 Overview of T3

1: construct the graph $\mathbf{W}$ from the raw datasets
2: compute the proximity matrices $\mathbf{ttP}$ and $\mathbf{toP}$
3: find time cluster membership function $\mathbf{g}$ based on $\mathbf{ttP}$
4: find the interpretation for each time cluster based on $\mathbf{toP}$

9.3.2 Compute the Proximity matrices

The key point in T3 is to construct two proximity matrices ($\mathbf{ttP}$ and $\mathbf{toP}$), based on which we will find the time cluster membership function $\mathbf{g}$ and its interpretations, respectively.

Alg. 18 lists detailed procedures to compute these two proximity matrices. Overall, we adopt the well-studied model of random walk with restart [HLZ+04, PYFD04, TFP06] for this purpose (steps 7-12). Suppose a random particle starts from the time node j, the particle iteratively transmits to its neighborhood with the probability that is proportional to the edge weight between them; and also at each step, it has some probability $(1-c)$ to return to the starting node j. The proximity score $r_{i,j}$ is defined as the steady-state probability that the particle will finally stay at node i. A subtle point in computing the proximity matrices is how to normalize the original adjacency matrix $\mathbf{W}$. In Alg. 18, we propose to normalize it by object type (steps 1-7). That is, suppose the random particle stays at some node of type x and overall there are s_x different types of objects connected to the x^{th} type of object; then at the next step, the particle will have equal chance ($\frac{1}{s_x}$) to jump to each of s_x types of objects.

9.3.3 Find Time Clusters

Here, we want to find the cluster membership function $\mathbf{g}$ for time stamps based on the time-to-time proximity matrix $\mathbf{ttP}$. The algorithm is listed in Alg. 19. We use a spectral clustering

Algorithm 18 Compute the Proximity Matrices $\mathbf{ttP}$ and $\mathbf{toP}$

Require: the adjacency matrix $\mathbf{W}$ and c
Ensure: the proximity matrices $\mathbf{ttP}$ and $\mathbf{toP}$
1: **for** $x = 1 : 2 + p + q$ **do**
2: **for** $y = 1 : 2 + p + q$ **do**
3: normalize by object type: $\mathbf{W}^{x,y} \leftarrow \frac{1}{s_x} \cdot (\mathbf{D}^{x,y})^{-1} \cdot \mathbf{W}^{x,y}$
4: **end for**
5: **end for**
6: set $\mathbf{W} \leftarrow [\mathbf{W}^{x,y}]$
7: **for** $j = 1 : n_1$ **do**
8: let $\mathbf{e} = \mathbf{0}_{n \times 1}$; then set $\mathbf{e}(j) = 1$
9: solve $\mathbf{r}$ from the equation $\mathbf{r} = c\mathbf{W}'\mathbf{r} + (1 - c)\mathbf{e}$
10: set $\mathbf{ttP}(:, j) = \mathbf{r}(1 : n_1)$
11: set $\mathbf{toP}(:, j) = \mathbf{r}(n_1 + 1 : n)$
12: **end for**

Algorithm 19 Find the Time Cluster

Require: the time-to-time proximity matrix $\mathbf{ttP}$
Ensure: the cluster membership function $\mathbf{g}$
1: do eigen value decomposition for $\mathbf{ttP}$; let $\{\lambda_1, ..., \lambda_{n_1}\}$ be the eigen values for $\mathbf{ttP}$ (from largest to smallest) and $\{\mathbf{v}_1, ..., \mathbf{v}_{n_1}\}$ be the corresponding eigen vectors
2: find the cluster number $z = \text{argmax}_i(\lambda_{i-1} - \lambda_i)$
3: let $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_z]$
4: treat each row of $\mathbf{V}$ as a data point in z-dimensional space
5: use k-means to find z clusters on $\mathbf{V}$ and output the corresponding cluster membership function $\mathbf{g}$

algorithm.[2] In Alg. 19, we first use the eigen-gap [**?**] (step 2) to choose cluster number z. Then, we treat the first z eigen vectors as the embedding of the time nodes in the z-dimensional space (steps 3-4) and run k-means to find the final cluster membership function $\mathbf{g}$ (step 5).

As mentioned before, if we find some cluster which contains a single time stamp, we flag it as the abnormal time stamp.

One benefit of using spectral clustering method is that we can use the first few eigen vectors as the embedding of the time stamps in some low dimensional space. For example, we can visualize the time stamps by plotting its first two eigen vectors in Fig. 9.1 for the running example.

[2]Notice that our framework is orthogonal to the specific clustering methods. We can plug in any clustering algorithm that takes a proximity matrix between nodes as input. For example, we could transfer the time-to-time proximity matrix $\mathbf{ttP}$ to be the normalized graph Laplacian and find its eigen-decomposition instead (step 1). Alternatively, we can normalize each row of $\mathbf{V}$ to have the unit length in step 3 as suggested in [NJW01].

9.3.4 Find Interpretations for Time Clusters

For each time cluster, we want to select a representative subset of instance nodes from each type of object (except the 'time' object) as the interpretations for that time cluster.

Suppose we want to find the interpretations for the time cluster u ($u = 1, ..., z$). Let $\bar{r}(j, u)$ be the average proximity score from the time cluster u to the instance node j:

$$\bar{r}(j, u) = \frac{\sum_{i=1}^{n_1} \mathrm{I}(\mathbf{g}(i) = u)\mathbf{toP}(j, i)}{\sum_{i=1}^{n_1} \mathrm{I}(\mathbf{g}(i) = u)} \tag{9.2}$$

where $\mathrm{I}(.)$ is an indicator function, which is 1 if the condition in the parenthesis is true and 0 otherwise.

Based on $\bar{r}(j, u)$, we can define the representative score $r(j, u)$ for each instance node j w.r.t. the given time cluster u as follows:

$$r(j, u) = \bar{r}(j, u) \prod_{w=1, w\neq u}^{z} (1 - \bar{r}(j, w)) \tag{9.3}$$

The intuition of Eq. (9.3) is that we want to find the node j which is close to the time cluster u (higher $\bar{r}(j, u)$ is better) and far away from other time clusters (lower $\bar{r}(j, w)(w \neq u)$ is better) on average. Finally, we can output a subset of instance nodes with high representative scores $r(j, u)$ from each type of object as the interpretations for the time cluster u.

9.4 MT3 for Multiple scale Analysis

In this section, we propose MT3 to address Problem 13. Conceptually, we can apply T3 for each scale of interest independently. Here, the challenge is to make the analysis on the coarser scales as efficient as possible, given that we have already done the analysis at the finest scale.

In Alg. 17, the computational bottleneck lies in step 2 – i.e., to compute the two proximity matrices $\mathbf{ttP}$ and $\mathbf{toP}$. For example, our experiments show that the time for this step usually accounts for more than 95% of the overall running time of the algorithm. Therefore, our goal in *Multiple Scale Analysis* is to efficiently update these two proximity matrices ($\tilde{\mathbf{ttP}}$ and $\tilde{\mathbf{toP}}$) at the aggregated scale, given that we have already computed the proximity matrices ($\mathbf{ttP}$ and $\mathbf{toP}$) at the finest scale.

We introduce the following vector $\mathbf{h}_{\mathbf{n_1}\times\mathbf{1}}$, where $\mathbf{h}(i) :=$ number of event/entity/attribute nodes connected to the time node i at the finest scale. Suppose that we will have $\tilde{n}_1$ time stamps at the aggregated scale (i.e., $\tilde{n}_1 = \max(\mathbf{f})$). Alg. 20 gives the detailed procedure to update the proximity matrices. In Alg. 20, after we get the overall normalized adjacency matrix $\tilde{\mathbf{W}}$ at the aggregated scale (step 1), we set up two transformation matrices $\mathbf{T}_1$ and $\mathbf{T}_2$ (steps 2-9). Then (steps 10-12), we need two matrix inversions (one $n_1 \times n_1$ in step 10 and one $\tilde{n}_1 \times \tilde{n}_1$ in step 11) to get the proximity matrices ($\tilde{\mathbf{ttP}}$ and $\tilde{\mathbf{toP}}$) at the aggregated scale. Note that in many real applications the number of time nodes at the finest scale is usually much smaller compared to the total nodes in the graph

Algorithm 20 Update the Proximity Matrices

Require: the proximity matrices $\mathbf{ttP}$ and $\mathbf{toP}$, the normalized adjacency matrix $\mathbf{W}$ at the finest scale, the aggregation function $\mathbf{f}$ and c;

Ensure: the proximity matrices $\tilde{\mathbf{ttP}}$ and $\tilde{\mathbf{toP}}$ at the aggregated scale.

1: set up the normalized adjacency matrix $\tilde{\mathbf{W}} = [\tilde{\mathbf{W}}^{x,y}]$ at the aggregated scale.
2: initialize the transformation matrices: $\mathbf{T}_1 = \mathbf{0}_{\tilde{n}_1 \times n_1}$, and $\mathbf{T}_2 = \mathbf{0}_{n_1 \times \tilde{n}_1}$
3: **for** $\tilde{i} = 1 : \tilde{n}_1$ **do**
4: find time stamps at the finest scale: $\mathcal{J} = \{i : \mathbf{g}(i) = \tilde{i}\}$
5: **for** each $i \in \mathcal{J}$ **do**
6: set $\mathbf{T}_1(\tilde{i}, i) = \mathbf{h}(i) / \sum_{i \in \mathcal{J}} \mathbf{h}(i)$
7: set $\mathbf{T}_2(i, \tilde{i}) = 1$
8: **end for**
9: **end for**
10: set $\mathbf{\Lambda} = \mathbf{I}_{n_1 \times n_1} - c\mathbf{W}'_{1,1} - (1-c)(\mathbf{ttP})^{-1}$
11: update $\tilde{\mathbf{ttP}} = (1-c)(\mathbf{I}_{\tilde{n}_1 \times \tilde{n}_1} - c\tilde{\mathbf{W}}'_{1,1} - \mathbf{T}'_2 \mathbf{\Lambda} \mathbf{T}'_1)^{-1}$
12: update $\tilde{\mathbf{toP}} = \mathbf{toP}(\mathbf{ttP})^{-1}\mathbf{T}'_1\tilde{\mathbf{ttP}}$

(i.e., $n_1 \ll n$). Typically, n_1 (the number of time nodes at the finest scale) is up to a few thousand whereas n (the total nodes in the graph) could be up to a few hundred thousand. For example, in the *DBLP* dataset, we only have about 49 among 988,947 time nodes at the finest scale. Therefore, we can efficiently update the proximity matrices at the aggregated scale by Alg. 20.

The correctness of Alg. 20 is guaranteed by the following theorem:

Theorem 10. *The proximity matrices* $\tilde{\mathbf{ttP}}$ *and* $\tilde{\mathbf{toP}}$ *by Alg. 20 are correct. That is, they are exactly the same as we apply Alg. 18 to the adjacency matrix* $\tilde{\mathbf{W}}$.

Proof. To simplify the description, we re-write the normalized adjacency matrix as the following 2×2 block form:

$$\mathbf{W} = \begin{pmatrix} \mathbf{A}^{1,1} & \mathbf{A}^{1,2} \\ \mathbf{A}^{2,1} & \mathbf{A}^{2,2} \end{pmatrix}, \quad \tilde{\mathbf{W}} = \begin{pmatrix} \tilde{\mathbf{A}}^{1,1} & \tilde{\mathbf{A}}^{1,2} \\ \tilde{\mathbf{A}}^{2,1} & \tilde{\mathbf{A}}^{2,2} \end{pmatrix} \tag{9.4}$$

where

$$\begin{aligned} \mathbf{A}^{1,1} &= \mathbf{W}^{1,1}, \ \tilde{\mathbf{A}}^{1,1} = \tilde{\mathbf{W}}^{1,1} \\ \mathbf{A}^{1,2} &= [\mathbf{W}^{1,y}], \ \tilde{\mathbf{A}}^{1,2} = [\tilde{\mathbf{W}}^{1,y}] \ (y = 2, ..., 2+p+q) \\ \mathbf{A}^{2,1} &= [\mathbf{W}^{x,1}], \ \tilde{\mathbf{A}}^{2,1} = [\tilde{\mathbf{W}}^{x,1}] \ (x = 2, ..., 2+p+q) \\ \mathbf{A}^{2,2} &= [\mathbf{W}^{x,y}], \ \tilde{\mathbf{A}}^{2,2} = [\tilde{\mathbf{W}}^{x,y}] \ (x, y = 2, ..., 2+p+q) \end{aligned} \tag{9.5}$$

Notice that only time nodes change before/after the aggregation, we have,

$$\tilde{\mathbf{A}}^{2,2} = \mathbf{A}^{2,2} \tag{9.6}$$

Furthermore, we can verify the following equations hold for the two off-diagonal blocks in Eq. (9.4):

$$\begin{aligned} \tilde{\mathbf{A}}^{1,2} &= \mathbf{T}_1 \mathbf{A}^{1,2} \\ \tilde{\mathbf{A}}^{2,1} &= \mathbf{A}^{2,1} \mathbf{T}_2 \end{aligned} \tag{9.7}$$

Define the following matrix inversion:

$$\begin{aligned} \mathbf{Q} &= (\mathbf{I} - c\mathbf{W})^{-1} \\ &= \begin{pmatrix} \mathbf{Q}^{1,1} & \mathbf{Q}^{1,2} \\ \mathbf{Q}^{2,1} & \mathbf{Q}^{2,2} \end{pmatrix} \\ \tilde{\mathbf{Q}} &= (\mathbf{I} - c\tilde{\mathbf{W}})^{-1} \\ &= \begin{pmatrix} \tilde{\mathbf{Q}}^{1,1} & \tilde{\mathbf{Q}}^{1,2} \\ \tilde{\mathbf{Q}}^{2,1} & \tilde{\mathbf{Q}}^{2,2} \end{pmatrix} \end{aligned} \tag{9.8}$$

By the property of random walk with restart [TFP06], we have the following equations for the proximity matrices:

$$\begin{aligned} \mathbf{ttP} &= (1-c)(\mathbf{Q}^{1,1})', \ \mathbf{toP} = (1-c)(\mathbf{Q}^{1,2})' \\ \tilde{\mathbf{tt}}\mathbf{P} &= (1-c)(\tilde{\mathbf{Q}}^{1,1})', \ \tilde{\mathbf{to}}\mathbf{P} = (1-c)(\tilde{\mathbf{Q}}^{1,2})' \end{aligned} \tag{9.9}$$

Now, apply block matrix inversion lemma [PC90] to Eq. (9.8). Together with Eq. (9.4)-(9.9), we have

$$\begin{aligned} \frac{1}{1-c}(\mathbf{ttP})' &= (\mathbf{I} - c\mathbf{W}^{1,1} - c^2 \mathbf{A}^{1,2}(I - \mathbf{A}^{2,2})^{-1}\mathbf{A}^{2,1})^{-1} \\ (\mathbf{toP})' &= c(\mathbf{ttP})'\mathbf{A}^{1,2}(I - \mathbf{A}^{2,2})^{-1} \\ \frac{1}{1-c}(\tilde{\mathbf{tt}}\mathbf{P})' &= (\mathbf{I} - c\tilde{\mathbf{W}}^{1,1} - c^2 \mathbf{T}_1\mathbf{A}^{1,2}(I - \mathbf{A}^{2,2})^{-1}\mathbf{A}^{2,1}\mathbf{T}_2)^{-1} \\ (\tilde{\mathbf{to}}\mathbf{P})' &= c(\tilde{\mathbf{tt}}\mathbf{P})'\mathbf{T}_1\mathbf{A}^{1,2}(I - \mathbf{A}^{2,2})^{-1} \end{aligned} \tag{9.10}$$

In Eq. (9.10), we have four equations for four unknown variables ($\tilde{\mathbf{tt}}\mathbf{P}$, $\tilde{\mathbf{to}}\mathbf{P}$, $\mathbf{A}^{1,2}(I-\mathbf{A}^{2,2})^{-1}\mathbf{A}^{2,1}$, and $\mathbf{A}^{1,2}(I - \mathbf{A}^{2,2})^{-1}$). Solving this well-defined linear system, we have

$$\begin{aligned} \tilde{\mathbf{tt}}\mathbf{P} &= (1-c)(\mathbf{I} - c\tilde{\mathbf{W}}'_{1,1} - \mathbf{T}'_2\mathbf{\Lambda}\mathbf{T}'_1)^{-1} \\ \tilde{\mathbf{to}}\mathbf{P} &= \mathbf{toP}(\mathbf{ttP})^{-1}\mathbf{T}'_1\tilde{\mathbf{tt}}\mathbf{P} \end{aligned} \tag{9.11}$$

where $\mathbf{\Lambda} = \mathbf{I} - c\mathbf{W}'_{1,1} - (1-c)(\mathbf{ttP})^{-1}$, which completes the proof of theorem 10. □

Based on Alg. 20, the complete algorithm for *Multiple Scale Analysis* is given in Alg. 21.

Algorithm 21 MT3 for Multiple Scale Analysis

Require: the proximity matrices $\mathbf{ttP}$ and $\mathbf{toP}$, the normalized adjacency matrix $\mathbf{W}$ at the finest scale, the aggregation function $\mathbf{f}$ and c

Ensure: (i) the cluster membership function $\tilde{\mathbf{g}}$ at the aggregated scale; and (ii) for each time cluster at aggregated scale, a representative subset of instances from each type of object (except 'time' object)

1: update the proximity matrices $\tilde{\mathbf{ttP}}$ and $\tilde{\mathbf{toP}}$ by Alg. 20
2: find the cluster membership function $\tilde{\mathbf{g}}$ by Alg. 19
3: for each time cluster $\tilde{u}$ in $\tilde{\mathbf{g}}$, compute the representative score $r(j, \tilde{u})$ for each instance j by $\tilde{\mathbf{toP}}$ and Eq. (9.3); and output a representative subset of instances from each type of object (except 'time' object) based on $r(j, \tilde{u})$

9.5 Experimental Results

In this section, we introduce four real data sets and present our experimental results. All of the experiments are designed to answer the following questions:

- *effectiveness:* What is the quality of T3 and MT3 proposed in this chapter?
- *efficiency:* How fast are the proposed algorithms?

9.5.1 Data Sets

Table 9.3: Datasets used in our evaluations

Dataset name	p	q	n_1	n	m
NIPS	1	0	13	3,900	11,460
CIKM	2	1	15	3,299	10,228
DBLP	2	0	49	988,947	5,216,722
DeviceScan	2	0	294	114,540	684,276

We use four real data sets, which are summarized in Table 3. For each data set, Table 9.5.1 lists the number of different types of 'entity' objects (p), the number of different types of 'attribute' objects (q), the number of time nodes in the finest scale (n_1), the number of nodes (n) and edges (m) in the whole graph in the finest scale. We verify the effectiveness of the proposed T3 and MT3 on *NIPS*, *CIKM*, and *DeviceScan*, and measure the efficiency of our algorithms using the larger *DBLP* and *DeviceScan* data sets.

The first data set (*NIPS*) is from the NIPS proceedings.[3] The time stamps are publication years, from 1987 to 1999. We treat paper as 'event' object and author as 'entity' object; there is no 'attribute' object in this data set. Overall, there are 13 time nodes, 1,740 paper nodes, 2,037 author nodes, and 11,460 edges at the finest scale.

[3] `http://www.cs.toronto.edu/~roweis/data.html`

The *CIKM* data set is constructed from the CIKM proceedings.[4] Again, time stamps are publication years, from 1993 to 2007. (Notice that we do not include papers from CIKM 1992 since the session information for that year is not available.) We treat paper as 'event' object. For this data set, we have two types of 'entity' objects: the authors of the paper and the session name where the paper is presented during the conference. For the session name, we further extract 158 keywords as its attribute. Overall, there are 15 time nodes, 952 paper nodes, 1,895 author nodes, 279 session nodes, 158 keyword nodes, and 10,228 edges at the finest scale.

The *DBLP* data set is constructed from all the papers in the DBLP.[5] Again, time stamps are publication years, from 1959 to 2007. We treat paper as 'event' object. For this data set, we have two types of 'entity' objects: the authors of the paper and the conference where the paper is published. There is no additional 'attribute' object for this data set. Overall, there are 49 time nodes, 567,090 paper nodes, 418,236 author nodes, 3,571 conference nodes, and 5,216,722 edges at the finest scale.

The *DeviceScan* is from MIT reality mining project.[6] Here, the 'event' object is blue tooth device scanning persons, and the time stamps are the day when such scanning events happen, from Jan. 1, 2004 to May. 5, 2005. For this data set, we have two types of 'entity' objects: the blue tooth device and the person to be scanned; there is no additional 'attribute' object. Overall, there are 294 time nodes, 114,046 scanning nodes, 103 device nodes, 97 person nodes, and 684,276 edges at the finest scale.

9.5.2 Effectiveness: Case Studies

Here, we show the experimental results for the three real data sets, all of which are consistent with our intuition.

Time Cluster	Selected Papers	Selected Authors
1987	Presynaptic Neural Information Processing	Sejnowski_T
1988	Phasor Neural Networks	Hinton_G
1989	Phase Transitions in Neural Networks	Mozer_M
1990	The Hopfield Model with Multi-Level Neurons	Moody_J
1991	Temporal Patterns of Activity in Neural Networks	Koch_C
	Mapping Classifier Systems Into Neural Networks	Jordan_M
1992	Digital Boltzmann VLSI for Constraint Satisfaction and Learning	Sejnowski_T
1993	Sparse Code Shrinkage, Denoising by Nonlinear Maximum	Jordan_M
1994	Likelihood Estimation	Hinton_G
1995	Efficient Approaches to Gaussian Process Classification,	Koch_C
1996	Image Representations for Facial Expression Coding	Williams_C
1997	Experiences with Bayesian Learning in a Real World Application	Dayan_P
1998	Spectral Cues in Human Sound Localization,	Maass_W
1999	A PolygonalLine Algorithm for Constructing Principal Curves	Sollich_P

Table 9.4: The interpretations for *NIPS* data set.

Fig. 9.2 gives the embedding of the time nodes for *NIPS* data set using the first two eigen

[4] http://www.informatik.uni-trier.de/~ley/db/conf/cikm/

[5] http://www.informatik.uni-trier.de/~ley/db/

[6] http://reality.media.mit.edu/

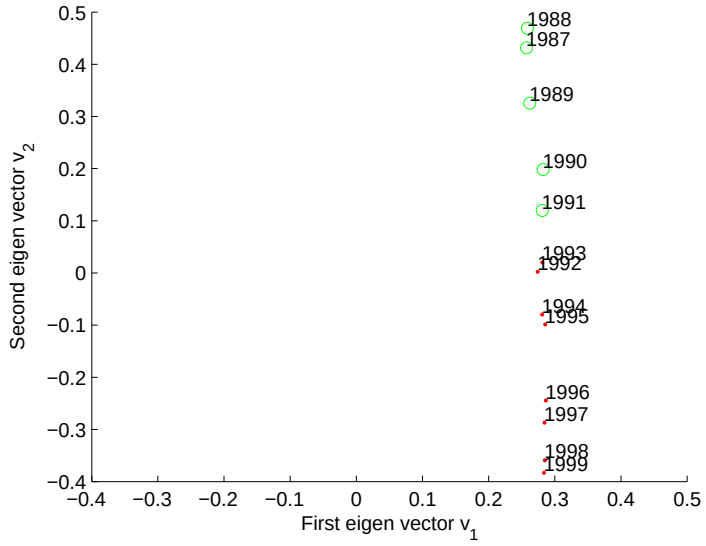


Figure 9.2: The embedding for the time nodes of *NIPS* data set.

vectors ($\mathbf{v}_1$ and $\mathbf{v}_2$) of $\mathbf{toP}$, which reveal a line shape of time over publication years. Using T3, we find two time clusters (green circles vs. red dots in Fig. 9.2) as well as their interpretations in Table 4. From Fig. 9.2 and Table 4, we can see that while NIPS is a relatively stable community on the whole (e.g., the majority representative authors do not change over years), there is a topic shift from early 1990s (mainly on 'neural network' and 'neural information processing') to late 1990s (mainly on 'statistical learning').

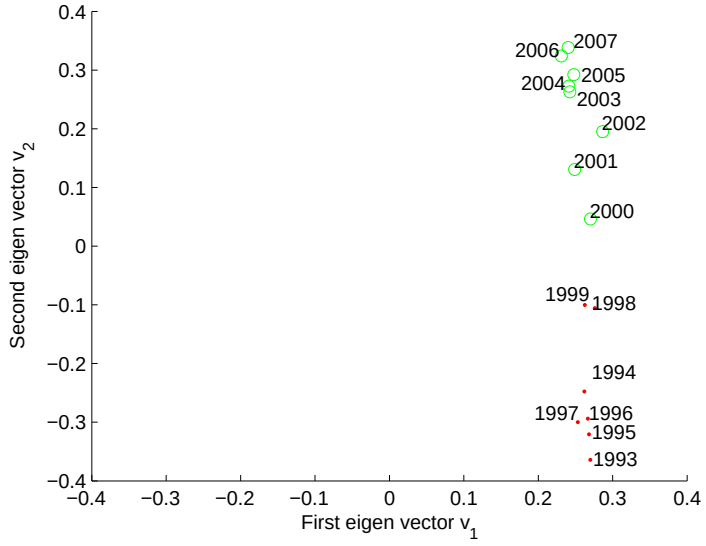


Figure 9.3: The embedding for the time nodes of *CIKM* data set.

Fig. 9.3 gives the embedding of the time nodes for *CIKM* data set using the first two eigen vectors ($\mathbf{v}_1$ and $\mathbf{v}_2$) of $\mathbf{toP}$, which reveal a line shape of time over publication years as for the *NIPS* data set. Using T3, we find two time clusters (green circles vs. red dots in Fig. 9.3) as well as their interpretations in Table 5. (For simplicity, we do not show the representative papers in the table.) From Fig. 9.3 and Table 5, we can see that while there are quite a lot of research interest in deductive databases and rule systems in the CIKM community in 1990s, attention has shifted to XML, statistical learning, language, etc since 2000.

Time Cluster	Selected Sessions	Selected Authors	Selected Keywords
	deductive_&_rule_systems	elke_rundensteiner	knowledge
1993	query_processing	daniel_miranker	system
1994	knowledge_representation_&_expert_systems	andreas_henrich	unstructured
1995	object-oriented_databases	il-yeol_song	rule
1996	access_to_unstructured_information	scott_b._huffman	transaction
1997	deductive_databases	ling_liu	object-oriented
1998	document_processing	robert_j._hall	document
1999	logical_&_deductive_databases	jian_tang	deductive
	tools_for_realizing_intelligent_systems	ibrahim_kamel	ai
2000	xml_schemas:_integration_and_translation	james_p._callan	web
2001	classification	javed_a._aslam	cluster
2002	language_models	w._bruce_croft	classification
2003	corpus_linguistics	marius_pasca	language
2004	high-dimensional_indexing	james_allan	xml
2005	semistructured_data	philip_s._yu	similarity
2006	data_warehousing_and_olap	anton_leuski	stream
2007	text_classification_and_categorization	george_karypis	learn
	summarization_and_corpus_analysis	charles_clarke	mobile

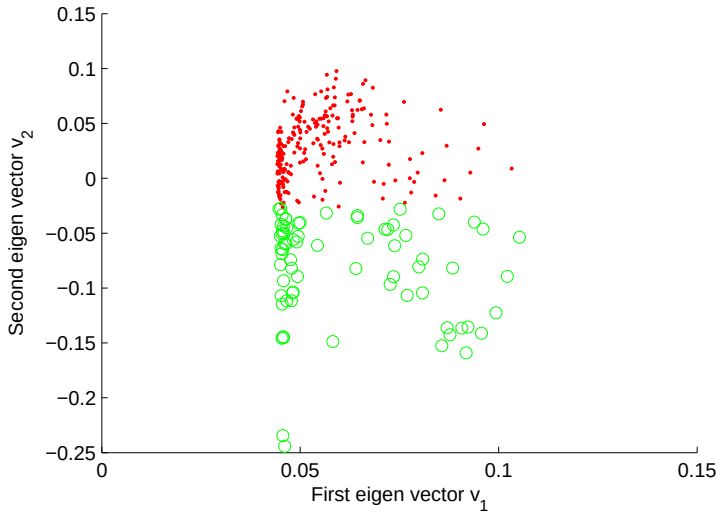
Table 9.5: The interpretation for *CIKM* data set.

Fig. 9.4 shows the results of applying the proposed MT3 to the *DeviceScan* data set on two different scales: (a) daily scale and (b) monthly scale. From Fig. 9.4(a), it can be seen that, there are two time clusters on the daily scale. We found that one time cluster (green circles) corresponds to semester breaks as well as holidays; and the other cluster (red dots) corresponds to the week days during the semester. On the other hand, we found an abnormal time stamp (red dot, which is Apr. 2004) on the monthly scale (Fig. 9.4(b)). This might be due to the spring break in Apr. 2004.

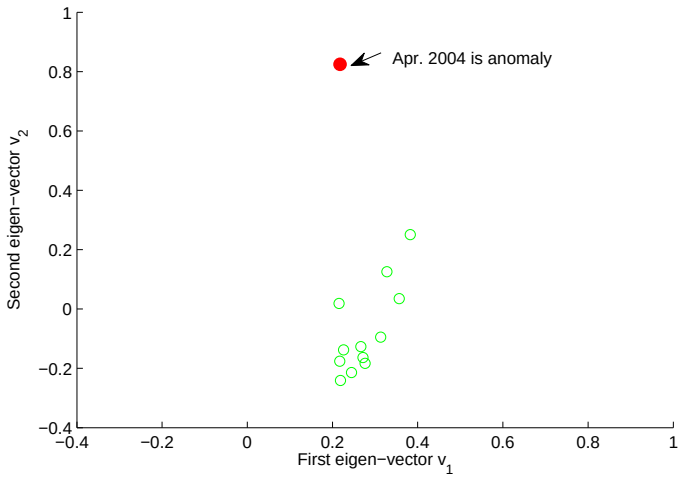
9.5.3 Efficiency

Here, we study the wall-clock time of the proposed MT3 using two relatively larger data sets: *DeviceScan* and *DBLP*. For these results, all of the experiments are done on the same machine with four 2.4GHz AMD CPUs and 48GB memory, running Linux (2.6 kernel). We vary the aggregation length (e.g., aggregate by every 2 time stamps, by every 3 time stamps, etc) and compare the wall-clock time by the proposed MT3 and that by applying T3 to each of the aggregated scale from scratch (referred to as the 'straight-forward' method).

Figure. 9.5 shows the results. Notice that time is in logarithm scale. It can be seen that the proposed MT3 is much more efficient. For example, it is 120x faster (6.1 seconds vs. 734 seconds) for *DeviceScan* data set if we aggregate the time by every three time stamps (Fig. 4.7(a)); and it is 263x faster (6.0 seconds vs. 1,603 seconds) for *DBLP* data set if we aggregate the time by every two time stamps (Fig. 4.7(b)). Overall, the proposed MT3 is 25x-263x faster than the straight-forward



(a) on daily scale



(b) on monthly scale

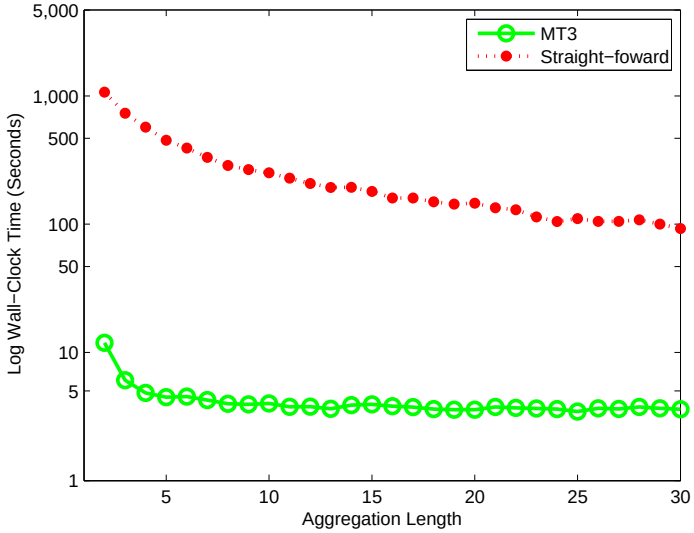
Figure 9.4: The embedding for the time nodes of *DeviceScan* data set.

method. We would like to emphasize that such speed-ups are totally *free*, i.e., the proposed MT3 leads to exactly the same outputs as we apply T3 to each aggregated scale from scratch.

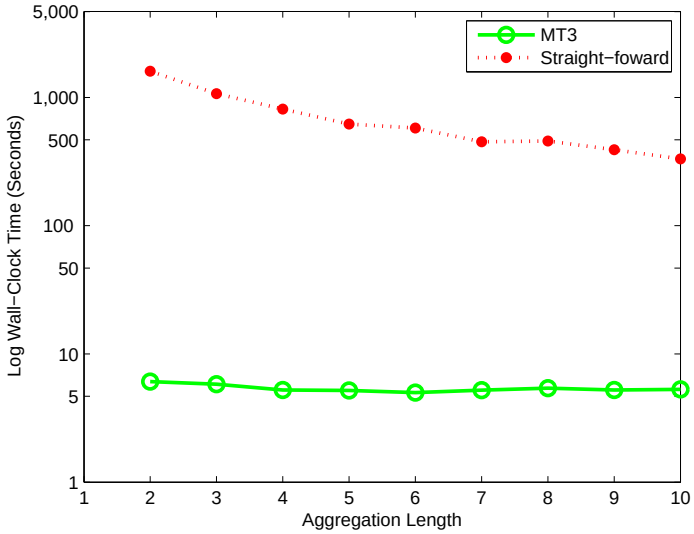
9.6 Related Work

In this section, we review the related work, which can be categorized into three parts: graph mining, proximity measurement on graphs and relational learning.

Graph Mining. There exists a lot of research on static graph mining (refer to Chapter 6 for detailed reviews). It is worth pointing out that in these work, the focus is on utilizing the time



(a) *DeviceScan* data set



(b) *DBLP* data set

Figure 9.5: Comparison on wall-clock time

information to better understand other nodes (event/entity/attribute) in the graphs; while in T3 and MT3 we focus on the other side of the problem, i.e., to better understand time itself based on other information (event/entity/attribute).

Measuring Proximity on Graphs. One of the most widely used proximity measurement on graphs is random walk with restart (refer to Chapter 2 for detailed reviews). Notice that the fast algorithms to compute the proximity measurements designed for querying, such as the one in [TFP06], do not apply in our settings since the pre-computational time for these algorithms will

flood the overall running time of T3 and MT3.

Also, there are a lot of applications of proximity measurements (again, refer to Chapter 2 for detailed reviews). The most related works are [PYFD04, BHP04, ACA06b, AC07] in the sense that they all use a graph representation for the dataset(s). However, these approaches mainly focus on querying with or without learning; while T3 and MT3 are focusing on mining time in the context of complicated events.

Relational Learning. Sharan and Neville [SN07] present a two-step approach for incorporating temporal information on links (e.g., co-authorship and citation) into a relational classifier. First, they summarize the time-varying interaction as weights on links of a static summary graph. The summarization uses an exponential weighting scheme [CPV01]. Second, they incorporate these link weights into a relational Bayes classifier. Their approach requires a summary parameter (θ), that needs to be either provided by the user or tuned by the learning algorithm. Furthermore, their approach cannot handle temporally-varying attributes. Our approach do not require a user-provided parameter and can handle time associated with any aspect of an event.

9.7 Conclusion

In this chapter, we study how to find patterns in a collection of time-stamped, complex events. Our main contributions are the following:

1. We propose to treat each time-stamp as a node in a carefully constructed graph. This opens the door for the vast arsenal of graph mining algorithms (PageRank, graph partitioning, proximity analysis, CenterPiece Subgraphs, etc). We show how the proposed T3 can automatically group the time stamps into meaningful clusters, spot anomalies, and provide interpretations.
2. We propose MT3 to handle multiple scale analysis, achieving up to *2 orders of magnitude* speedups.
3. Finally, we verify the effectiveness as well as the efficiency of T3 and MT3 with experiments on several real datasets.

A promising research direction is to extend the T3 and MT3 to include additional continuous attributes, like geographical coordinates.

Part VI

Conclusion and future directions

Chapter 10

Conclusion and Future Work

10.1 Summary of Contributions

Graphs appear in a wide range of settings and have posed a wealth of fascinating problems. According to the interaction with users, the research focus of this thesis work lies in two parts: (1) querying and (2) mining. The main contributions of the thesis can be summarized as follows:

Querying Graphs:

- **Complex User-Specific Patterns.** We found that many complex user-specific patterns on large graphs can be answered by means of proximity measurement. In other words, *proximity allows us to query large graphs on the atomic levels*. We support our claim by conducting three case studies (center-piece subgraphs (chapter 3), user feedback (chapter 4), and gateway (chapter 5)), all of which (despite the difference in applications) rely on proximity measurement as their building block.

  Impact/Results. The proposed algorithms are operational, with careful design and numerous optimizations. They led to 3 patents pending. The proposed algorithms for both *CePS* and user feedback are to be deployed into a real product (*Cyano*) in IBM [QSJ08].
- **Proximity Tracking.** We proposed an efficient algorithm *pTrack* (chapter 6) to track proximity on time-evolving graphs.

  Impact/Results. It enables us to do trend analysis on the graph level. The proposed algorithm (*pTrack*) is up to *176x* faster than competitors and has no quality loss (Theorem 4). This work won the *Best Paper* award in SIAM-DM 2008.
- **Fast Proximity Computations.** We developed a family of fast solutions (*FastRWR*) (chapters 2-6,9) to compute proximity in several different scenarios. The idea is to carefully leverage some important properties shared by many real graphs (e.g., the block-wise structure, the linear correlation, the skewness of real bipartite graphs, etc)

  Impact/Results. We can often achieve orders of magnitude (*up to 6,000,000x*) speedup with little (e.g., Theorem 1, Lemma 6, etc) or no quality loss (e.g., Lemma 2, Theorem 10, etc). One of these works [TFP06] won the *Best Research Paper* award in ICDM 2006.

Mining Graphs:

- **Vulnerability Analysis.** We proposed an algorithm NetShield (chapter 6) for immunization under the SIS (susceptible-infection-susceptible) model.
  Impact/Results. While straight-forward methods are computationally intractable ($O(\binom{n}{k}m)$), the proposed algorithm is *near-optimal* (Theorem 5), *fast* (up to 7 orders of magnitude speedup), and *scalable* ($O(nk^2 + m)$) .
- **Anomaly Detection.** We proposed a family of example-based low-rank matrix approximation methods *Colibri* (chapter 7) for anomaly detection.
  Impact/Results. The proposed algorithms are provably equal to or better than the best known methods with respect to both space and time (e.g., Lemma 17, etc), with the same accuracy (e.g., Theorem 6, Lemma 18, etc). On real data sets, it is up to *112x* faster than the best competitors, for the same accuracy.
- **Mining Complex Time-Stamped Events.** We show that graphs also provide a very powerful tool to solve some complex problems. As a case study (chapter 9), we proposed a general framework (*T3*) to mine complex time stamped events, by formulating the problem as a graph analysis problem. We further proposed MT3 to handle multiple-scale analysis.
  Impact/Results. The proposed *T3* is able to find similar time stamps, find abnormal time stamps and provide interpretations for our findings. The proposed MT3 achieves up to *2 orders of magnitude* speedup, with the same quality (Theorem 10).

10.2 Vision for the Future

In the thesis, we show that graphs provide a very powerful and unified tool to handle data heterogeneity, with an intuitive user interface. On themselves, graphs pose a wealth of fascinating research questions and high-impact applications. It is my belief that graphs will continue to play an even more important role in our lives, - more and more real applications will rely on graphs; much richer types of graphs will show up; and the scales of real-world graphs will continue to grow.

My long-term research goal is to help the user to better *understand* and *utilize* large real graph data sets. More specifically, there are three closely related dimensions of this research goal:

G1. (**Querying**) Given a graph (say, a social network), how to help the user to find things according to his/her particular interest?
G2. (**Mining**) Given a graph, how to succinctly describe it, and report anomalies?
G3. (**Scalability**) How to scale our querying and mining algorithms to large graphs, spanning multiple machines?

Along the way to fulfill this research goal, our research focus will span on the following five aspects, which are separated in medium term goal (M1-M3) and long term goal (L1-L2).

M1. Design new algorithms for recommendations on large graphs.
M2. Design new algorithms for immunization.

M3. Improve the usability of graph querying and mining results, by giving interpretation and summarization of querying and mining results.

L1. Address the scalability issue.

L2. Address rich types of data, specifically weighted graphs, attributed graphs, time-evolving graphs, and geo-coded graphs.

Answering these questions are critical for many high-impact applications. Among others, our motivating applications are:

- (Social Networks) Effective querying and recommendation tools are playing an important role in on-line social network sites, - with hundreds of millions of users [LH08].
- (Security) Graph querying algorithms can help to find suspicious subgraphs (e.g., master-mind criminal in law enforcement [CAW+06], money-laundering ring in financial fraud [MBA+09], suspicious communication patterns, etc).
- (Epidemiology) A good immunization strategy might help to prevent an epidemic from out-breaking with the lowest cost [CWW+07].
- (E-commerce/Viral Marketing) A good immunization strategy can also help to spot the 'best' customers for advertisement ('k-advertisement') in viral marketing, which can largely improve the revenue [DR01].
- (Communication networks) Graph mining algorithms can help to detect abnormal behaviors in both computer networks and phone networks (e.g., port scanning, router mis-configuration, telemarketing, etc)

The relationship between these applications and our long term research goal is summarized in table 10.1.

Table 10.1: Applications of Long Term Research Goals

	Social Networks	**Security**	**Epidemiology**	**E-commerce**	**Communication Networks**
G1: Querying	✓	✓			
G2: Mining	✓		✓	✓	✓
G3: Scalability	✓	✓	✓	✓	✓

In the thesis, we have made the first step towards such long term goal. For example, we have designed several algorithms to find complex user-specific patterns on large graphs. For the SIS (susceptible-infection-susceptible) model, we have designed a near-optimal immunization strategy. We can detect one specific type of anomaly (linear correlation) from large graphs, using our *Colibri*. We have shown that graphs usually provide a friendly user interface, and that example-based methods are promising for interpreting the mining results. For all the algorithms we proposed in the thesis, they are scalable (linear with respect to the size of the graph or better).

Next, we will present our medium term plan and long term plan, respectively. These steps, and their relationship with the thesis work, are summarized in table 10.2.

Table 10.2: Vision for the Future

Goals \ Plans	Step 1 (thesis work)	Step 2 (medium term)	Step 3 (long term)
G1 (Querying)	Chapter 2-6	Recommendation Interpretable querying	Querying rich types of data
G2 (Mining)	Chapter 7-9	Immunization Interpretable mining	Mining rich types of data
G3 (Scalability)	Chapter 2-9	O(E) or better (a single machine)	Scalable on Map-Reduce Scalable on rich types of data

10.2.1 Medium Term Plan

In the near future, we will focus on the following three tasks, all of which are built on the thesis work:

M1: Broad Spectrum Recommendation Systems

A large portion of the thesis work focuses on querying large graphs. In other words, if the user knows what s/he exactly wants, we are now in a better position in helping them to find such things (e.g., center-piece subgraphs, gateway, etc). In the next step, we would like to help the user to find things that s/he might not (or partially) know, where recommendation plays a crucial role.

While most of the existing work focuses on relevance (i.e., find things that are most relevant to the user's interest), there are other important aspects in recommendation, e.g., novelty, diversity etc. For example, our preliminary work in [OTF09] shows that by taking into account the novelty in recommendation, we can broaden user's horizon.

Here, our ultimate goal is to provide the user a subset of items which covers the broad spectrum of his/her interest (e.g., relevance, diversity and novelty). In order to achieve this goal, we need to work on *'broad spectrum recommendation'*, where we aim to *collectively* find the whole recommended subset, instead of a list of *individual* items.

M2: Immunization

In the thesis, we have designed a very promising immunization algorithm for SIS (susceptible-infection-susceptible) model. We will generalize our work to (1) immunize under other types of virus propagation models (e.g., SIR (susceptible-infection-recovery), or the mixture of SIS and SIR, etc); (2) immunize in the case the graph structure is changing over time).

M3: Interpretation of Querying and Mining Results

Most real data sets do not have labels. It usually takes a lot of time for the analyst to check/understand the mining results. Therefore, it is important to generate a concise and intuitive explanation for the user to better understand the mining results. In the thesis (chapter 8), we show that a few representative examples are usually very helpful to interpret the querying and mining results (e.g., communities, anomalies, etc).

We will continue on this line of research to further improve the usability of mining results. Here, the two main research questions we will address are (1) how to select a few examples/nodes as 'basis'; (2) how to use the selected examples to interpret the remaining nodes (e.g., by a sparse nonnegative linear combination).

10.2.2 Long Term Plan

In the long run, we will focus on the following two directions, all of which are common to both G1 (querying) and G2 (mining):

L1: Scalability

As the scale of the real data continues to grow, scalability is a 'never-ending' question in large graph mining. Here, we will deal with this issue through the following two orthogonal efforts: (1) continue to design scalable (linear or better) algorithms on a single machine; (2) explore map-reduce type abstractions for large scale computation on graphs, where the challenge is how to de-couple the computation among different machines.

L2: Rich Types of Graph Data

Most existing algorithms work on plain undirected graphs. We plan to extend our work to graphs with attributes (both on nodes and edges), time-evolving graphs, directed weighted graphs. As the main tool for analyzing single plain graphs is matrix algebra, in order to extend our algorithms to such types of graphs, we need to simultaneously analyze multiple inter-correlated matrices or to analyze tensor (the generalization of matrices). On the other hand, although graphs account for a large portion of real data sets, there are other types of data sets (e.g., spatial, temporal, etc). In the thesis (chapter 9), we show that we can handle complex time-stamped events by envisioning the problem as a graph analysis problem. We will continue on this line of research. Ideally, we would like to develop a unified model to handle such complex data (the mixture of relational, temporal and spatial data).

Bibliography

[ABC+02] B. Aditya, Gaurav Bhalotia, Soumen Chakrabarti, Arvind Hulgeri, Charuta Nakhe, and S. Sudarshan Parag. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, 2002.

[AC07] Alekh Agarwal and Soumen Chakrabarti. Learning random walks to rank nodes in graphs. In *ICML*, pages 9–16, 2007.

[ACA06a] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD*, pages 14–23, 2006.

[ACA06b] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD*, pages 14–23, 2006.

[AJB99] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Diameter of the world wide web. *Nature*, (401):130–131, 1999.

[AM01] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximation. In *STOC*, 2001.

[AM07] Dimitris Achlioptas and Frank McSherry. Fast computation of low-rank matrix approximations. *J. ACM*, 54(2), 2007.

[BHKL06] Lars Backstrom, Daniel P. Huttenlocher, Jon M. Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.

[BHP04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[BKM+00] Andrei Broder, Ravi Kumar, Farzin Maghoul1, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web: experiments and models. In *WWW Conf.*, 2000.

[CAW+06] Hsinchun Chen, Homa Atabakhsh, Alan Gang Wang, Siddharth Kaza, Chunju Tseng, Yuan Wang, Shailesh Joshi, Tim Petersen, and Chuck Violette. Coplink center: social network analysis and identity deception detection for law enforcement and homeland security intelligence and security informatics: a crime data mining approach to developing border safe research. In *DG.O*, pages 49–50, 2006.

[Cha05] Deepayan Chakrabarti. Tools for large graph mining. In *Carnegie Mellon University Technical Report*, 2005.

[CHbA91] Reuven Cohen1, Shlomo Havlin1, and Daniel ben Avraham. Efficient immunization strategies for computer networks and populations. *Phys. Rev. Lett.*, 91(24), 1991.

[CLR90] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[CPV01] C. Cortes, D. Pregibon, and C. Volinsky. Communities of interest. In *Proc. of the 4th Int'l Symp. of Intelligent Data Analysis (IDA'01)*, pages 105–114, 2001.

[CSZ+07] Yun Chi, Xiaodan Song, Dengyong Zhou, Koji Hino, and Belle L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD*, pages 153–162, 2007.

[CTSP07] Haibin Cheng, Pang-Ning Tan, Jon Sticklen, and William F. Punch. Recommendation via query centered random walk on k-partite graph. In *ICDM*, pages 457–462, 2007.

[CWW+07] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jure Leskovec, and Christos Faloutsos. Epidemic thresholds in real networks. *ACM Transactions on Information and System Security (ACM TISSEC)*, 10(4), 2007.

[DDL+90] S.C. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[DKM05a] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM Journal of Computing*, 2005.

[DKM05b] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM Journal of Computing*, 2005.

[DLJ08] Chris H. Q. Ding, Tao Li, and Michael I. Jordan. Nonnegative matrix factorization for combinatorial optimization: Spectral clustering, graph matching, and clique finding. In *ICDM*, pages 183–192, 2008.

[DM02] S.N. Dorogovtsev and J.F.F. Mendes. Evolution of networks. *Advances in Physics*, 51:1079–1187, 2002.

[DMM03] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-theoretic co-clustering. In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 03)*, Washington, DC, August 24-27 2003.

[DMM07] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error cur matrix decompositions. *CoRR*, abs/0708.3696, 2007.

[DR01] Pedro Domingos and Matthew Richardson. Mining the network value of customers. In *KDD*, pages 57–66, 2001.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *SIGCOMM*, pages 251–262, Aug-Sept. 1999.

[Fie73] M. Fiedler. Algebraic connectivity of graphs. 1973.

[FLG00] G.W. Flake, S. Lawrence, and C.L. Giles. Efficient identification of web communities. In *KDD*, pages 150–160, 2000.

[FLGC02] Gary Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3), March 2002.

[FMT04] Christos Faloutsos, Kevin S. McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *KDD*, pages 118–127, 2004.

[FR04] D. Fogaras and B. Racz. Towards scaling fully personalized pagerank. In *Proc. WAW*, pages 105–117, 2004.

[Fre77] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[GGK03] Sudipto Guha, Dimitrios Gunopulos, and Nick Koudas. Correlating synchronous and asynchronous data streams. In *KDD*, pages 529–534, 2003.

[GKR98] David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *Ninth ACM Conference on Hypertext and Hypermedia*, pages 225–234, New York, 1998.

[GKRT04] Ramanathan V. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *WWW*, pages 403–412, 2004.

[GL96] G.H. Golub and C.F.V. Loan. *Matrix Computation*. Johns Hopkins, 1996.

[GMS93] 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.

[GMT04] Floris Geerts, Heikki Mannila, and Evimaria Terzi. Relational link-based ranking. In *VLDB*, pages 552–563, 2004.

[GMZ03] Christos Gkantsidis, Milena Mihail, and Ellen W. Zegura. Spectral analysis of internet topologies. In *INFOCOM*, 2003.

[GN] Michelle Girvan and M. E. J. Newman. Community structure is social and biological networks.

[GVL89] G. H. Golub and C. F. Van-Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[Hav02] Taher H. Haveliwala. Topic-sensitive pagerank. *WWW*, pages 517–526, 2002.

[Hav03] Taher H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.

[HLZ+04] Jingrui He, Mingjing Li, Hong-Jiang Zhang, Hanghang Tong, and Changshui Zhang. Manifold-ranking based image retrieval. In *ACM Multimedia*, pages 9–16, 2004.

[Ind00] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS*, pages 189–197, 2000.

[Jol02] I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[JW02] Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

[JW03] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, 2003.

[KAS98] Kothuri Venkata Ravi Kanth, Divyakant Agrawal, and Ambuj K. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD Conference*, pages 166–176, 1998.

[KG05] Andreas Krause and Carlos Guestrin. Near-optimal value of information in graphical models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2005.

[KG07] Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, pages 1650–1654, 2007.

[KHMG03] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub. Exploiting the block structure of the web for computing pagerank. In *Stanford University Technical Report*, 2003.

[KK99] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.

[KKT03] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. *KDD*, 2003.

[Kle98] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[KNV06] Yehuda Koren, Stephen C. North, and Chris Volinsky. Measuring and extracting proximity in networks. In *KDD*, pages 245–255, 2006.

[Koz92] Dexter C. Kozen. *The Design and Analysis Algorithms*. Springer-Verlag, 1992.

[LH08] Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *WWW*, pages 915–924, 2008.

[LJM+07] Wangzhong Lu, Jeannette C. M. Janssen, Evangelos E. Milios, Nathalie Japkowicz, and Yongzheng Zhang. Node similarity in the citation graph. *Journal Knowledge and Information Systems*, 11(1):105–129, 2007.

[LKF05] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[LNK03] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proc. CIKM*, 2003.

[LTL03] Chin Lung Lu, Chuan Yi Tang, and Richard Chia-Tung Lee. The steiner tree problem. *Theoretical Computer Science*, 306:55–67, 2003.

[MBA+09] Mary McGlohon, Stephen Bay, Markus G. Anderle, David M. Steier, and Christos Faloutsos. Snare: a link analytic system for graph labeling and risk detection. In *KDD*, pages 1265–1274, 2009.

[MC06] Einat Minkov and William W. Cohen. An email and meeting assistant using graph walks. In *CEAS*, 2006.

[MW08] J. Ian Munro and Dorothea Wagner. Better approximation of betweenness centrality. 2008.

[New03] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[New05] M.E.J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27:39–54, 2005.

[NJW01] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856, 2001.

[NXC+07] Huazhong Ning, Wei Xu, Yun Chi, Yihong Gong, and Thomas S. Huang. Incremental spectral clustering with application to monitoring of evolving blog communities. In *SDM*, 2007.

[OTF09] Kensuke Onuma, Hanghang Tong, and Christos Faloutsos. Tangent: a novel, 'surprise me', recommendation algorithm. In *KDD*, pages 657–666, 2009.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry 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).

[PC90] W.W. Piegorsch and G. Erratum Casella. Inverting a sum of matrices. In *SIAM Review*, volume 32, pages 470–470, 1990.

[PF03] Christopher R. Palmer and Christos Faloutsos. Electricity based external similarity of categorical attributes. *PAKDD 2003*, April-May 2003.

[PRTU05] Luigi Palopoli, Domenico Rosaci, Giorgio Terracina, and Domenico Ursino. A graph-based approach for extracting terminological properties from information sources with heterogeneous formats. *Journal Knowledge and Information Systems*, 8(4):462–497, 2005.

[PSF05] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

[PYFD04] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[QSJ08] Huiming Qu, Jimeng Sun, and Hani Jamjoom. Scoop: Automated social recommendation in enterprise process management. In *IEEE SCC (1)*, pages 101–108, 2008.

[RMPS05] Cartic Ramakrishnan, William Milnor, Matthew Perry, and Amit Sheth. Discovering informative connection subgraphs in multi-relational graphs. *SIGKDD Explorations Special Issue on Link Mining*, 2005.

[RW06] Carl Edward Rasmusen and Chris Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[SFPY07] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD*, pages 687–696, 2007.

[SM97] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *CVPR*, pages 731–737, 1997.

[SN07] Umang Sharan and Jennifer Neville. Exploiting time-varying relationships in statistical relational models. In *1st SNA-KDD Workshop*, 2007.

[SQCF05] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.

[SS90] G. W. Stewart and Ji-Guang Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[STF06] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD*, pages 374–383, 2006.

[SXZF07] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *SDM*, 2007.

[TF06] Hanghang Tong and Christos Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[TFGER07] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.

[TFP06] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[TFP08] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Random walk with restart: Fast solutions and applications. *Knowledge and Information Systems: An International Journal (KAIS)*, 2008.

[TKF07] Hanghang Tong, Yehuda Koren, and Christos Faloutsos. Fast direction-aware proximity for graph mining. In *KDD*, pages 747–756, 2007.

[TPYF08] Hanghang Tong, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, pages 704–715, 2008.

[TQJF09] Hanghang Tong, Huiming Qu, Hani Jamjoom, and Christos Faloutsos. ipog: Fast interactive proximity querying on graphs. In *CIKM*, 2009.

[Tso08] Charalampos E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM*, pages 608–617, 2008.

[XHYC05] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *VLDB*, pages 709–720, 2005.

[Zac77] W. W. Zachary. An information flow model for conflict and fission in small groups. pages 452–473, 1977.

[ZBL+03] D. Zhou, O. Bousquet, T.N. Lal, J. Weston, and B. Scholkopf. Learning with local and global consistency. In *NIPS*, 2003.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, pages 912–919, 2003.

[ZHD+01] Hongyuan Zha, Xiaofeng He, Chris H. Q. Ding, Ming Gu, and Horst D. Simon. Spectral relaxation for k-means clustering. In *NIPS*, pages 1057–1064, 2001.

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of, "Don't ask, don't tell, don't pursue," excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213, telephone (412) 268-2056

Obtain general information about Carnegie Mellon University by calling (412) 268-2000