Connectivity in Complex Networks:

Measures, Inference and Optimization

by

Chen Chen

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved March 2019 by the
Graduate Supervisory Committee:

Hanghang Tong, Chair
Hasan Davulcu
Arunabha Sen
V.S. Subrahmanian
Lei Ying

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Networks naturally appear in many high-impact applications. The simplest model of networks is single-layered networks, where the nodes are from the same domain and the links are of the same type. However, as the world is highly coupled, nodes from different application domains tend to be interdependent on each other, forming a more complex network model called multi-layered networks.

Among the various aspects of network studies, network connectivity plays an important role in a myriad of applications. The diversified application areas have spurred numerous connectivity measures, each designed for some specific tasks. Although effective in their own fields, none of the connectivity measures is generally applicable to all the tasks. Moreover, existing connectivity measures are predominantly based on single-layered networks, with few attempts made on multi-layered networks.

Most connectivity analyzing methods assume that the input network is static and accurate, which is not realistic in many applications. As real-world networks are evolving, their connectivity scores would vary by time as well, making it imperative to keep track of those changing parameters in a timely manner. Furthermore, as the observed links in the input network may be inaccurate due to noise and incomplete data sources, it is crucial to infer a more accurate network structure to better approximate its connectivity scores.

The ultimate goal of connectivity studies is to optimize the connectivity scores via manipulating the network structures. For most complex measures, the hardness of the optimization problem still remains unknown. Meanwhile, current optimization methods are mainly ad-hoc solutions for specific types of connectivity measures on single-layered networks. No optimization framework has ever been proposed to tackle a wider range of connectivity measures on complex networks.

In this thesis, an in-depth study of connectivity measures, inference, and optimiza-

tion problems will be proposed. Specifically, a unified connectivity measure model will be introduced to unveil the commonality among existing connectivity measures. For the connectivity inference aspect, an effective network inference method and connectivity tracking framework will be described. Last, a generalized optimization framework will be built to address the connectivity minimization/maximization problems on both single-layered and multi-layered networks.

This dissertation is dedicated to my beloved parents Jinxia Xu and Yuelong Chen.

ACKNOWLEDGMENTS

First, I would like to express my deep gratitude to my advisor Dr. Hanghang Tong for his guidance, patience, inspiration and great support. Five years ago, I was just a layman to the data mining field and felt great uncertainty to my upcoming Ph.D. study. Nourished by the unreserved help and advising from Dr. Tong, I started with my baby steps by learning to write good papers and make good presentations; and then began to make progress through discovering and conquering interesting research problems. The past five years have revolutionized my life as my knowledge boundary has been greatly expanded from time after time. Meanwhile, the encouragement and support from Dr. Tong have also empowered me with the strong will to fight against frustrations and the persistence to pursue tough goals.

I would also like to thank my committee members Dr. Hasan Davulcu, Dr. Arunabha Sen, Dr. V.S. Subrahmanian and Dr. Lei Ying for their insightful comments on my research. Specifically, I would like to thank Dr. Ying for being a great collaborator and sharing with me his sharp understanding of network connectivity.

It was a great experience to work as an Intern in Futurewei Technologies Inc. with amazing mentors and colleges: Yinglong Xia, Hui Zang, Jiangsheng Yu, Zonghuan Wu, Qingsong Wen, Yiming Kong, Tingyi Zhu, Ting Yang, Li Zhou, and Cheng Bo. The diversity of the group has given me a great chance to learn from experts in different domains and discover inspirational ideas. Special thank to Dr. Xia, who was my mentor during the internship, for introducing me the challenges on large-scale graph computation platforms.

As a member of Data and Star Labs, I am grateful for the consistent support and encouragement from all other members: Liangyue Li, Xing Su, Si Zhang, Boxin Du, Qinghai Zhou, Jian Kang, Zhe Xu, Lihui Liu, Scott Freitas, Haichao Yu, Ruiyue Peng, Rongyu Lin, Xiaoyu Zhang, Dawei Zhou, Yao Zhou, Arun Reddy, Xu Liu, Xue

Hu, Jun Wu, Lecheng Zheng, Dongqi Fu, Pei Yang, and Qi Tan. In addition, I would like to thank all my friends for your company. You are all imperative parts of this amazing journey.

Finally, I am so grateful to have the best parents, grandparents and the whole family in the world. Your unconditional love, support, and patience is the motivation for me to become a better me. Last but not least, I would like to thank my boyfriend Jundong Li for standing by me through all the thick and thin.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

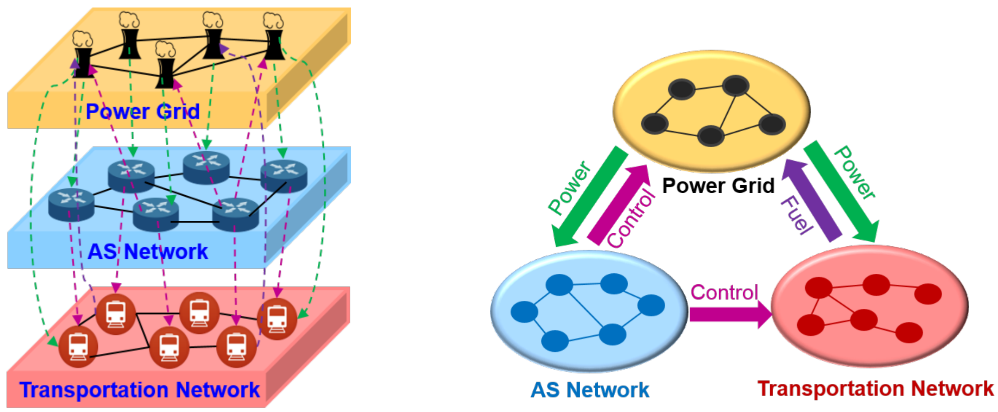
Figure Page

Figure Page

Chapter 1

INTRODUCTION

Networks are prevalent in many high-impact domains, including information dissemination, social collaboration, infrastructure constructions, and many more. The most well-studied type of networks is single-layered networks, where the nodes are collected from the same domain and the links are used to represent the same type of connections. However, as the world is becoming highly connected, cross-domain interactions are more frequently observed in numerous applications, catalyzing the emergence of a new network model–*multi-layered networks* Buldyrev *et al.* (2010); Gao *et al.* (2012); Parshani *et al.* (2010); Sen *et al.* (2014). One typical example of such type of network is the critical infrastructure network as illustrated in Figure 4.25. In an infrastructure network system, the full functioning of the autonomous system network (AS network) and the transportation network is dependent on the power supply from the power grid. While for the gas-fired and coal-fired generators in the power grid, their functioning is fully dependent on the gas and coal supply from the transportation network. Moreover, to keep the whole complex system working in order, extensive communications are needed between the nodes in the networks, which are supported by the AS network. In addition to the infrastructure systems, multi-layered networks also appear in many other application domains, such as organization-level collaboration platform Chen *et al.* (2015) and cross-platform e-commerce systems Chen *et al.* (2013); Li *et al.* (2009); Lu *et al.* (2013); Yang *et al.* (2015).



(a) Critical Infrastructure Network (b) Dependency Relations across the Layers

Figure 1.1: An Illustrative Example of Multi-layered Networks. In Figure 4.25(b), Each Ellipse Corresponds to a Critical Infrastructure Network in Figure 4.25(a) (I.E., Power Grid, as Network and Transportation Network). The Arrows Between Two Ellipses Indicate Cross-layer Dependency Relationships Between the Corresponding Two Networks (E.G., A Router in the as Network Depends on One or More Power Plants in the Power Grid).

1.1 Research Challenges

Connectivity Measures. Among the various network properties studied in the literature, network connectivity is the one that plays a crucial role in applications like disease control, network robustness analysis, community detection, etc. Correspondingly, different connectivity measures are designed for each of the applications. Examples include epidemic threshold Chakrabarti *et al.* (2008) for disease dissemination analysis, natural connectivity Jun *et al.* (2010) for robustness measurement and triangle capacity for social network mining. Empirical analysis has demonstrated the effectiveness of those connectivity measures in their own tasks, but none of them can be used as a common measure across different domains. Furthermore, most, if not all, of the existing connectivity measures are defined on single-layered networks, leaving the problem of measuring multi-layered network connectivity unexplored.

Connectivity Inference. Existing network connectivity research predominantly assumes that the input network is static and accurate, which does not fit into the dynamic and noisy real-world settings. Real-world networks are evolving over time. In some cases, subtle changes in the network structure may lead to huge differences on some of the connectivity measures. For example, in websites like Facebook and Twitter, new connections between users emerge all the time, which would, in turn, change the influential individuals in the network. Thus, it is crucial for online marketing companies to keep track of those changes since their advertisements targeting strategies may need to be modified accordingly. On the other hand, in multi-layered networks, it remains a daunting task to know the exact cross-layer dependency structure due to noise, incomplete data sources and limited accessibility issues. For example, an extreme weather event might significantly disrupt the power grid, the transportation network and the cross-layer dependencies in between at the epicenter. Yet, due to limited accessibility to the damaged area during or soon after the disruption, the cross-layer dependency structure might only have a probabilistic and/or coarse-grained description.

Connectivity Optimization. The crucial task for network connectivity studies is to optimize (minimize/maximize) the connectivity score by adjusting the underlying network structure. Previous literature has proved that the optimization problem on epidemic threshold and triangle capacity on single-layered networks is NP-hard. However, for some complex connectivity measures (e.g., natural connectivity), the hardness of the corresponding optimization problems still remains unknown. Most importantly, existing connectivity optimization methods are mainly based on single-layered networks. Compared to single-layered networks, multi-layered networks are more sensitive to disturbance since its effect may be amplified through cross-layer dependencies in all the dependent networks, leading to a cascade failure of the entire system. To tackle the connectivity optimization problem in multi-layered networks, great efforts have been made from different research area for manipulating *two-layered* interdependent network systems Buldyrev *et al.* (2010); Parshani *et al.* (2010); Sen *et al.* (2014); Gao *et al.* (2012). Although much progress has been made, challenges are still largely open. First, as the connectivity measures are highly diversified, the ad-hoc optimization algorithms that are effective for specific measures may not work well on other measures. Thus, the problem of how to design a generic optimization strategy for a wide range of network connectivity measures is in need of investigation. Second, existing optimization strategies tailored for two-layered networks might be sub-optimal, or even misleading for arbitrarily structured multi-layered networks, because it can not effectively unravel the nested dependency structure in the network.

1.2 Tasks Overview

The main problems studied in this thesis are focused on measures, inference, and optimization of network connectivity in complex networks. The relationship between those problems is shown in Figure 1.2. Generally speaking, a well defined connectivity measure serves as the objective to inference and optimization tasks; The inference results, in turn, provide a good approximation on the connectivity measure and improve the accuracy of the input network for optimization tasks; Last, the optimization methods are used to find optimal strategies to manipulate the network structure, which can effectively change the connectivity of the network and influence the inference results from task 2.

1.3 Organization

The remainder of the thesis is organized as follows. In Chapter 2, I will review the related literature on network connectivity measures, inference methods, and optimization algorithms. In chapter 3, I will introduce the generalized definition for

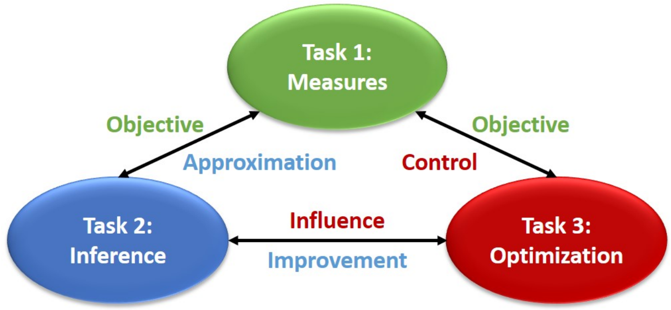


Figure 1.2: Tasks Overview.

connectivity measures in single-layered networks and its extension on multi-layered networks. In Chapter 4, I will address the eigen-functions (connectivity) tracking problem in dynamic networks and the cross-layer dependence inference problem in multi-layered networks. In Chapter 5, I will study the fundamental limits and efficient algorithms for the network connectivity optimization problem in single-layered networks and its extension on multi-layered networks. Finally, in Chapter 6, I will conclude the thesis and discuss future work.

Chapter 2

LITERATURE REVIEW

In this section, we organize the related research about network connectivity in complex networks into four parts: (1) multi-layered network modeling; (2) network connectivity measures; (3) network connectivity inference; (4) network connectivity optimization and (5) network connectivity applications.

2.1 Multi-layered Network Modeling

Multi-layered networks have attracted a lot of research attention in recent years. Different models have been proposed to formulate the multi-layered network data structure. In De Domenico *et al.* (2013), multi-layered networks are represented as a high-order tensor, which is coupled by a second-order within-layer networks tensor and a second-order cross-layer dependency tensor. While in Sánchez-García *et al.* (2014), the corresponding data structure is represented as a quadruplet $M = \{V_M, E_M, V, \mathcal{L}\}$, in which each distinct nodes in V can appear in multiple elementary layers in $\mathcal{L} = \{L_1, \ldots, L_d\}$. Then, $V_M \subseteq V \times L_1 \times \ldots \times L_d$ represents the nodes in each layer, and $E_M = V_M \times V_M$ represents both within-layer and cross-layer links in the entire system. In Boccaletti *et al.* (2014), the model is simplified into a pair $M = (\mathcal{G}, \mathcal{C})$, where $\mathcal{G}$ gives all the within-layer networks and $\mathcal{C}$ provides all the cross-layer dependencies. Different from the above models, the formulation used in our study gives more emphasis on the abstracted dependency network structure $\mathbf{G}$, which makes it easier to unravel the impact path for a set of nodes from a given layer. In Kivelä *et al.* (2014), Kivela et al. presented a comprehensive survey on different types of multi-layered networks, which include multi-modal networks Heath and Sioson (2009), multi-dimensional networks Berlingerio *et al.* (2011), multiplex networks Battiston *et al.* (2014) and interdependent networks Buldyrev *et al.* (2010). The problem addressed in our study is most related to the interdependent networks. In Rinaldi *et al.* (2001) and Gao *et al.* (2011), the authors presented an in-depth introduction to the fundamental concepts of interdependent multi-layered networks as well as the key research challenges. In a multi-layered network, the failure of a small number of nodes might lead to catastrophic damages on the entire system as shown in Buldyrev *et al.* (2010) and Vespignani (2010). In Buldyrev *et al.* (2010); Parshani *et al.* (2010); Shao *et al.* (2011); Sen *et al.* (2014); Gao *et al.* (2012), different types of *two-layered* interdependent networks were thoroughly analyzed. In Gao *et al.* (2011), Gao et al. analyzed the robustness of multi-layered networks with the star- and loop-shaped dependency structures. In De Domenico *et al.* (2015), De Domenico et al. proposed a method to identify versatile nodes in multi-layered networks by evaluating their eigenvector centrality and PageRank centrality. The selected versatile nodes are fundamentally different from our high impact nodes for network connectivity in three aspects. First is that their centrality measures can not capture the collective impact of a node set on the network. Second is that our proposed network connectivity is directly related to only within-layer links, while cross-layer dependency is the trigger for connectivity changes. The two types of links should be treated differently rather than mixed up for a unified centrality calculation. Last, the globally crucial nodes in the entire system may not be able to provide an optimal solution to minimize the connectivity in specific target layer(s).

2.2 Network Connectivity Measures

Connectivity is a fundamental property of networks and has been a core research theme in graph theory and mining for decades [1] . At the macro-level, network connectivity can be viewed as a measure to evaluate how well the nodes are connected together. Depending on the specific applications, many network connectivity measures have been proposed in the past. Examples include the size of giant connected component (GCC), graph diameter, the mixing time Jerrum and Sinclair (1988), the vulnerability measure Albert *et al.* (2000), and the clustering coefficient Wasserman (1994), each of which often has its own, different mathematical definitions. At the micro-view level, network connectivity measures the capacity of edges, paths, loops, some complex motifs Milo *et al.* (2002) or even the centrality of the nodes. Some well-known examples from this category include the epidemic threshold Chakrabarti *et al.* (2008), the natural connectivity (i.e., the robustness) Jun *et al.* (2010), degree centrality Freeman (1978), etc.

2.3 Network Connectivity Inference

The research on network connectivity inference can be categorized into two parts: (1) network property tracking in single-layered networks and (2) inference in multi-layered networks.

Various network properties have been studied under dynamic settings. In Leskovec *et al.* (2007b), Leskovec et al. discovered the growth pattern of network density and diameter in real networks. In Tong *et al.* (2008), two online algorithms were provided

[1]In this thesis, 'graph' and 'network' are interchangeably equivalent.

for tracking node proximity and centrality on bipartite graphs. In Li *et al.* (2015b), a graph kernel tracking algorithm was proposed for dynamic networks. The other area of research that is related to our work is evolutionary spectral clustering on graphs. In Ning *et al.* (2010), Ning et al. proposed an incremental spectral clustering algorithm based on iterative update on the eigen-system of the network. In Chen and Tong (2017), Chen et al. proposed an efficient online algorithm to track some important network connectivity measures (e.g., the leading eigenvalue, the robustness measure) on a temporal dynamic network. For the multi-layered network inference problem, a collaborative filtering based method is proposed to infer the missing cross-layer dependencies in Chen *et al.* (2016c). Other remotely related studies include cross-network ranking Ni *et al.* (2014) in multi-layered networks, and multi-view data analysis Li *et al.* (2016); Xu *et al.* (2013); Zhou *et al.* (2015).

2.4 Network Connectivity Optimization

From the algorithmic perspective, network connectivity optimization aims to maximize or minimize the corresponding connectivity measures by manipulating the underlying topology (e.g., add/remove nodes/edges). Earlier work, either explicitly or implicitly, assumes that nodes/edges with higher centrality scores would have a greater impact on network connectivity. This assumption has led to many research efforts on finding good node/edge centrality measures (or node/edge importance measure in general). Some widely used centrality measures include shortest path based centrality Freeman (1977), PageRank Page *et al.* (1998), HITS Kleinberg (1998), coreness score Moody and White (2003), local Fiedler vector centrality Chen and Hero (2014) and random walks based centrality Newman (2005). Different from those node centrality oriented methods, some recent work aims to take one step further by *collectively* finding a subset of nodes/edges with the highest impact on the network connectivity measure. For example, Tong *et al.* (2010, 2012); Chen *et al.* (2016b,a) proposed both node-level and edge-level manipulation strategies to optimize the leading eigenvalue of the network, which is the key network connectivity measure for a variety of cascading models. In Chan *et al.* (2014), Chan et al. further generalized these strategies to manipulate the network robustness scores Jun *et al.* (2010). In Chen *et al.* (2018), Chen et al. showed that the connectivity optimization problem is generally NP-hard and proposed an efficient algorithm for both node and edge level optimization tasks.

2.5 Network Connectivity Applications

The connectivity of the network has played an important role in many applications. For the applications that focus on system-level studies, global connectivity measures are more commonly used. In the immunization studies Cohen *et al.* (2003), it is critical to select a group of nodes/edges to effectively contain the propagation process Chen *et al.* (2016b); Chen and Tong (2017). In the biomedical domain, antibiotic drugs are developed to kill the bacteria by disrupting their molecular network to the max extend Kohanski *et al.* (2010). While in the critical infrastructure networks, facilities that may cause large-scale failures are retrieved and protected proactively to ensure the full-functioning of the entire system Chen *et al.* (2017). It is worth to note that finding a group of nodes/edges that have high impact on the connectivity of the network is similar to the influence maximization problem Kempe *et al.* (2003); Morone and Makse (2015); Chen *et al.* (2009) and its variations (e.g., viral marketing Leskovec *et al.* (2007a); Chen *et al.* (2010), outbreak detection Leskovec *et al.* (2007c), etc). The main difference between the two problems is that the influence maximization problem is highly dependent on the underlying diffusion model (e.g., independent cascade, linear threshold Kempe *et al.* (2003)), while the connectivity optimization problem is directly based on the backbone (i.e., the underlying topology) of the network. In addition to the global connectivity measures, the local connectivity of the network has also been applied to some critical tasks like graph clustering Yin *et al.* (2017, 2019), link prediction Benson *et al.* (2018), etc.

Chapter 3

NETWORK CONNECTIVITY MEASURES

Various connectivity measures have been designed for different applications. In our work, we propose two unified frameworks to evaluate the connectivity in both single-layered networks and multi-layered networks.

3.1 Single-Layered Network Measures

In this section, we study the connectivity measures in single-layered networks from two perspectives.

Eigen-Function Based Connectivity Measures

In this section, we first introduce the definition of eigen-function based connectivity measures, and then give several prevalently used examples in the category.

The main idea here to define the connectivity of the network as a function of its eigen-pairs. Formally, let $\mathbf{\Lambda}, \mathbf{U}$ be the eigenvalue and eigenvector matrices of network G, then the connectivity of G can be defined as

$$C(G, g) = g(\mathbf{\Lambda}, \mathbf{U}) \tag{3.1}$$

where $g : (\mathbf{\Lambda}, \mathbf{U}) \rightarrow \mathbb{R}^{+}$ is a function that maps the eigen-pair of the network to a non-negative connectivity attribute or attribute vector.

Important Eigen-Functions:

Eigenvalues and Eigenvectors. Since the eigen-pairs of a graph are important attributes themselves, the simplest eigen-function is therefore an identity function as follows:

$$g(\mathbf{\Lambda}, \mathbf{U}) = (\mathbf{\Lambda}, \mathbf{U}) \tag{3.2}$$

The eigenvalues of a graph's adjacency matrix can be used to measure the epidemic threshold of a network Harary and Schwenk (1979), while the eigenvectors can be used to evaluate the centrality of nodes Newman (2008), or to detect interesting subgraphs Prakash *et al.* (2010).

Triangle Capacity. The number of triangles in a graph plays an important role in calculating clustering coefficient and related attributes. The brute-force algorithm for solving this problem is of complexity $O(n^3)$. The state-of-the-art algorithm has reduced the complexity to $O(n^{2.373})$ Williams (2011), but this is still not a scalable algorithm on real-world large datasets. In Tsourakakis (2008), Tsourakakis showed that the number of triangles in a graph($\triangle(G)$) can be calculated using Eq. (3.3).

$$g(\mathbf{\Lambda}, \mathbf{U}) = \triangle(G) = \frac{1}{6}\sum_{i=1}^{n} \lambda_i^3 \tag{3.3}$$

By Eq. (3.3), the triangle capacity $\triangle(G)$ therefore becomes a function of eigenvalues. In real-world graphs, usually, we only need top k eigenvalues to achieve a good approximation for triangle counting. For example, experiments in Tsourakakis (2008) showed that picking top 30 eigen-pairs can achieve an accuracy of at least 95% in most graphs.

Natural Connectivity. The natural connectivity of a network evaluates its tolerance under random failure and external attacks. The definition of natural connectivity ($S(G)$) Chan *et al.* (2014) is shown in Eq. (3.4).

$$g(\mathbf{\Lambda}, \mathbf{U}) = S(G) = \ln(\frac{1}{n}\sum_{i=1}^{n} e^{\lambda_i}) \tag{3.4}$$

Once again, in Chan *et al.* (2014), Chan et al. found that top k ($k = 50$ in their study) eigen-pairs are sufficient for estimating the connectivity score.

Eigen-Gap. The eigen-gap of a graph is an important parameter in expander graph theory and is defined as the difference between the largest and second largest (in module) eigenvalues of the graph (as shown in Eq. (3.5)).

$$g(\mathbf{\Lambda}, \mathbf{U}) = Gap(G) = \lambda_1 - \lambda_2 \tag{3.5}$$

In expander graph theory, a graph is considered to have a good expansion property if it is both sparse and highly connected Hoory *et al.* (2006). By Cheeger inequality, the expansion property of a graph is strongly correlated to its eigen-gap Chung (1997). As a result, the eigen-gap of the graph can be used as a measurement for its robustness.

SubLine Connectivity Measures

Here, we introduce another perspective to model network connectivity.

The key of SubLine is to view the connectivity of the entire network as the aggregation over the connectivity measures of its sub-networks (e.g., subgraphs), that is

$$C(G, f) = \sum_{\pi \subseteq G} f(\pi) \tag{3.6}$$

where π is a subgraph of G. The non-negative function $f : \pi \rightarrow \mathbb{R}^+$ maps any subgraph in G to a non-negative real number and $f(\phi) = 0$ for empty set ϕ. In other words, we view the connectivity of the entire network ($C(G, f)$) as the sum of the connectivity of all the valid subgraphs ($f(\pi)$). The definition in Eq. (3.6) can be used to measure the connectivity of the *entire* network. It can be further extended to measure the local connectivity of a subset of nodes $\mathcal{T}$, where we define $f(\pi) > 0$ iff π is incident to the node set $\mathcal{T}$, i.e.,

$$C_{\mathcal{T}}(G, f) = \sum_{\pi \cap \mathcal{T} \neq \phi} f(\pi) \tag{3.7}$$

It is worth to mention that motifs (defined in Milo *et al.* (2002)) are subgraphs as well. By setting function f as non-negative constants, many eigen-function based network connectivity measures can be reduced to SubLine connectivity measures; and we give three prominent examples below, including (1) the path capacity; (2) the triangle capacity and (3) the loop capacity.

Path Capacity. A natural way to measure network connectivity is through *path capacity*, which measures the total number of (weighted) paths in the network. In this case, the corresponding function $f()$ can be defined as follows.

$$f(\pi) = \begin{cases} \beta^{len(\pi)} & \text{if } \pi \text{ is a valid path of length } len(\pi) \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

where β is a damping factor between $(0, 1/\lambda_G)$ to penalize longer paths. With such an $f()$ function , the connectivity function $C(G, f)$ defined in Eq. (3.6) can be written as

$$C(G, f) = \mathbf{1}'(\sum_{t=1}^{\infty} \beta^t \mathbf{A}^t)\mathbf{1} = \mathbf{1}'(\mathbf{I} - \beta\mathbf{A})^{-1}\mathbf{1} \tag{3.9}$$

Remarks. We can also define the path capacity with respect to a given path length t as $C(G, f) = \mathbf{1}'\mathbf{A}^t\mathbf{1}$. When $t = 1$, $C(G, f)$ is reduced to the edge capacity (density) of the graph, which is an important metric for network analysis. On the other hand, the 'average' path capacity $(\mathbf{1}'\mathbf{A}^t\mathbf{1})^{1/t}$ of a network converges to the leading eigenvalue of its adjacency matrix, i.e., $(\mathbf{1}'\mathbf{A}^t\mathbf{1})^{1/t} \xrightarrow{t\to\infty} \lambda_G$, which is exactly an eigen-function based connectivity measure—epidemic threshold Tong *et al.* (2010).

Triangle Capacity. By the definition of SubLine connectivity, triangle capacity can be modeled by setting the function $f()$ as

$$f(\pi) = \begin{cases} 1 & \text{if } \pi \text{ is a triangle} \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

Eq. (3.3) indicates that triangle capacity can be calculated as an eigen-function as well.

Loop Capacity. Another important way to measure network connectivity is through the loop capacity, which measures the total number of (weighted) loops in the network. In this case, the corresponding function $f()$ can be defined as follows.

$$f(\pi) = \begin{cases} 1/len(\pi)! & \text{if } \pi \text{ is a valid loop of length } len(\pi) \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

Then, the connectivity function $C(G, f)$ can be written as

$$C(G, f) = \sum_{t=1}^{\infty} \frac{1}{t!} trace(\mathbf{A}^t) = \sum_{i=1}^{n} e^{\lambda_{<\mathbf{A},i>}} \tag{3.12}$$

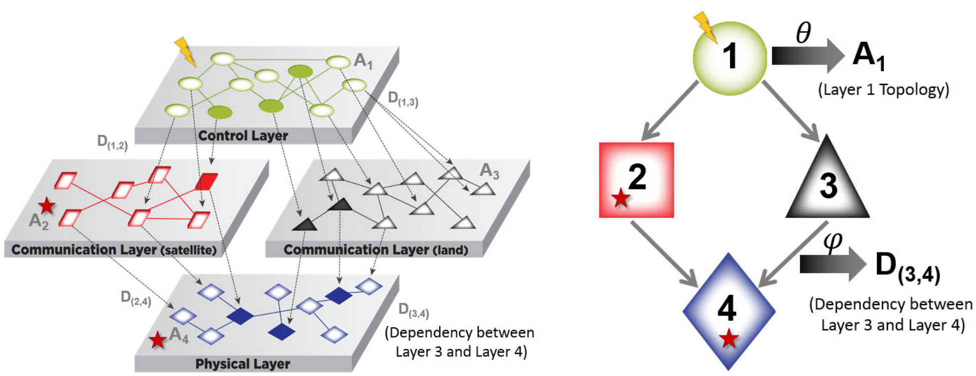
It is clear to see that the natural connectivity in the previous section is equivalent to the loop capacity in SubLine family where we have $S(G) = \ln(\frac{1}{n}C(G, f))$.

3.2 Multi-layered Network Measures

To extend the connectivity measures in single-layered networks to multi-layered networks, we first give the formal definition of multi-layered networks as follows.

Definition 1. ***Multi-layered Network Model (*****MuLaN*****)****.* *Given (1) a binary $g \times g$ abstract layer-layer dependency network $\mathbf{G}$, where $\mathbf{G}(i, j) = 1$ indicates layer-j depends on layer-i (or layer-i supports layer-j), $\mathbf{G}(i, j) = 0$ means that there is no direct dependency from layer-i to layer-j; (2) a set of within-layer networks $\{G_i\}_{i=1}^{g}$ with adjacency matrices $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_g\}$; (3) a set of cross-layer node-node dependency matrices $\mathcal{D}$, indexed by pair (i, j), $i, j \in [1, \ldots, g]$, such that for a pair (i, j), if $\mathbf{G}(i, j) = 1$, then $\mathbf{D}_{i,j}$ is a $n_i \times n_j$ matrix; otherwise $\mathbf{D}_{i,j} = \phi$ (i.e., an empty set); (4) θ is a one-to-one mapping function that maps each node in layer-layer dependency* *network* $\mathbf{G}$ *to the corresponding within-layer adjacency matrix* $\mathbf{A}_i$ $(i = 1, ..., g)$*; (5)* φ *is another one-to-one mapping function that maps each edge in* $\mathbf{G}$ *to the corresponding cross-layer node-node dependency matrix* $\mathbf{D}_{i,j}$*. We define a multi-layered network as a quintuple* $\Gamma = < \mathbf{G}, \mathcal{A}, \mathcal{D}, \theta, \varphi >$.

For simplicity, we restrict the within-layer adjacency matrices $\mathbf{A}_i$ to be simple (i.e., no self-loops), symmetric and binary; and the extension to the weighted, asymmetric case is straightforward. In our work, we require cross-layer dependency network $\mathbf{G}$ to be an un-weighted graph with arbitrary dependency structure. Notice that compared with the existing pair-wise two-layered models, MuLan allows a much more flexible and complicated dependency structure among different layers. For the cross-layer node-node dependency matrix $\mathbf{D}_{i,j}$, $\mathbf{D}_{i,j}(s,t) = 1$ indicates that node s in layer i supports node t in layer j.



(a) A four-layered network (b) The dependency network $\mathbf{G}$

Figure 3.1: An Illustrative Example Of MuLan model

Figure 3.1(a) presents an example of a four-layered network. In this example, layer-1 (i.e., the control layer) is the supporting layer (i.e., the root node in the layer-layer dependency network $\mathbf{G}$). Layer-2 and layer-3 directly depend on layer-1 (i.e., one represents a communication layer by satellites and the other represents another communication layer in landlines, respectively), while layer-4 (i.e., the physical layer) depends on both communication layers (layer-2 and layer-3). The abstracted layer-layer dependency network ($\mathbf{G}$) is shown in Figure 3.1(b). $\mathcal{A} = \{\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4\}$ denotes the within-layer adjacency matrices, each of which describes the network topology in the corresponding layer. In this example, $\mathcal{D}$ is a set of matrices containing only four non-empty matrices: $\mathbf{D}_{(1,2)}$, $\mathbf{D}_{(1,3)}$, $\mathbf{D}_{(2,4)}$, and $\mathbf{D}_{(3,4)}$. For example, $\mathbf{D}_{(3,4)}$ describes the node-node dependency between layer-3 and layer-4. The one-to-one mapping function θ maps node 1 in $\mathbf{G}$ (i.e., Layer 1) to the within-layer adjacency matrix of layer-1 ($\mathbf{A}_1$); and the one-to-one mapping function φ maps edge $\langle 3, 4 \rangle$ in $\mathbf{G}$ to the cross-layer node-node dependency matrix $\mathbf{D}_{(3,4)}$ as shown in Figure 3.1(b).

With the above definition, the connectivity of a multi-layered network Γ can be defined as a weighted summation over the connectivity scores from all the layers

$$C(\Gamma) = \sum_{i=1}^{g} \alpha_i C(G_i, f_i) \tag{3.13}$$

where α_i is the weight for layer i.

Chapter 4

NETWORK CONNECTIVITY INFERENCE

In real-world applications, networks are evolving over time. Moreover, the input network structure is often incomplete due to noise and accessibility issues. To address those issues, we propose to study the following two problems: (1) eigen-function based connectivity measures tracking in dynamic networks, and (2) cross-layer dependence inference in multi-layered networks.

4.1 Eigen-functions Tracking in Dynamic Networks

To better understand the node centrality and connectivity of graphs, many different graph parameters have been invented for different tasks. Though different in their definitions, many of those parameters can be well approximated by some well-defined eigen-functions. For example, in node centrality analysis, one commonly used parameter is eigenvector centrality Newman (2008), which is defined with the leading eigenvector of the graph. As for graph connectivity, frequently used parameters include epidemic threshold (Wang *et al.* (2003); Chakrabarti *et al.* (2008); Prakash *et al.* (2012)), clustering coefficient Wasserman (1994), graph robustness (Albert *et al.* (2000); Frank and Frisch (1970); Chan *et al.* (2014)), eigen-gap, etc. For epidemic threshold, Prakash et al. found that the tipping point for the dissemination process in *arbitrary* graphs is controlled by the leading eigenvalue of certain system matrix associated with the graph Prakash *et al.* (2012). For clustering coefficient calculation, the most time-consuming part is counting the number of triangles in the graph, which is of $O(n^3)$ complexity. In Tsourakakis (2008), Tsourakakis proved that the number of triangles in a graph can be accurately estimated with its top eigenvalues. Similar to clustering coefficient, Chan et al. showed that natural connectivity Wu *et al.* (2010), a good measurement for graph robustness, can also be approximated with the top eigenvalues of the graph. Moreover, as shown in Chung (1997), the expansion property of a graph can be measured with its eigen-gap between first and second eigenvalues Hoory *et al.* (2006).

Most of the graph parameters mentioned above are all based on static graphs. However, in real-world applications, the graph structure evolves over time. In some cases, subtle changes in the graph structure may lead to huge differences on some of its properties. For example, when Ebola virus was first brought to the US continent, some emerging connections in the contact network would greatly reduce the epidemic threshold of the graph, and eventually cause the outbreak of the disease. By monitoring those key parameters as graph evolves and analyzing the attribution for sharp parameter changes timely, we would be able to get prepared for emergent events at an early stage. Another application scenario is the social network. In websites like Facebook and Twitter, new connections between users emerge all the time, which would, in turn, change the influential individuals in the network. It is crucial for online marketing companies to keep track of those changes since their ads targeting strategies may need to be modified accordingly.

For eigen-functions tracking problem, simply re-computing the eigen-pairs whenever the graph structure changs is computationally costly over fast-changing large graphs. The popular Lanczos method for computing top-k eigen-pairs would require $O(mk + nk^2)$ time, where m and n are the numbers of edges and nodes in the graph. Although the complexity seems acceptable for one-time calculation in static graphs, it would be too expensive for large dynamic graphs. To address this challenge, we consider a way of updating the eigen-pairs incrementally instead of re-computing them from scratch at each time stamp. In this work, we propose two online algorithms to track the eigen-pairs of a dynamic graph efficiently, which bear linear time complexities w.r.t. the number of nodes n in the graph and the number of changed edges s at the current stamp. Based on these algorithms, we introduce a general attribution analysis framework for identifying key connection changes that have the largest impact on the graph. Last, to control the accumulated tracking error of eigen-functions, we propose an error estimation method to detect sharp error increase timely so that the accumulated error can be eliminated by restarting the tracking algorithms.

In addition to the problem definition, the main contributions of this work can be summarized as follows:

- *Algorithms.* We propose two online algorithms to track the top eigen-pairs of a dynamic graph, which in turn enable us to track a variety of important network parameters based on certain eigen-functions. In addition, we provide a framework for attribution analysis on eigen-functions and a method for estimating tracking errors.
- *Evaluations.* We evaluate our methods with other eigen-pair update algorithms on real-world datasets, to validate the effectiveness and efficiency of the proposed algorithms.

4.1.1 Problem Definition

In this section, we introduce the notations, followed by a formal definition of eigen-functions tracking problem.

The symbols used in this work is shown in Table 5.3. We consider the graph in each time stamp $G^t(V, E)$ is undirected and unipartite. Consistent with standard notation, we use bold upper-case for matrices (e.g., $\mathbf{B}$), and bold lower-case for vectors (e.g., $\mathbf{b}$). For each time stamp, the graph is represented by its adjacency matrix $\mathbf{A}^t$.

$\Delta\mathbf{A}^t$ denotes the perturbation matrix from time t to $t+1$. $(\lambda_j^t, \mathbf{u_j}^t)$ is the j^{th} eigen-pair of $\mathbf{A}^t$. The number of triangles and robustness score of the graph at time t are represented as $\triangle(G^t)$ and $S(G^t)$, respectively.

With the above notations, the eigen-function is defined as a function that maps eigen-pairs of the graph to certain graph attribute or attribute vector, which can be expressed as

$$g : (\Lambda_k, \mathbf{U}_k) \rightarrow \mathbb{R}^x (x \in \mathbb{N}) \tag{4.1}$$

Table 4.1: Symbols used in Trip-Basic and Trip.

Symbol	Definition and Description
$G^t(V, E)$	undirected, unipartite network at time t
m	number of edges in the network
n	number of nodes in the network
$\mathbf{B}, \mathbf{C}, \ldots$	matrices (bold upper case)
$\mathbf{b}, \mathbf{c}, \ldots$	vectors (bold lower case)
$\mathbf{A}^t$	adjacency matrix of $G^t(V, E)$ at time t
$\Delta\mathbf{A^t}$	perturbation matrix from time t to $t+1$
$\triangle(G^t)$	number of triangles in G^t
$S(G^t)$	robustness score of G^t
$Gap(G^t)$	eigen-gap of G^t
$(\lambda_j{}^t, \mathbf{u_j}^t)$	j^{th} eigen-pair of $\mathbf{A}^t$
$[\Delta\mathbf{A}^t]_{t=t_1 \ldots t_2}$	perturbation matrices of dynamic graph from time t_1 to t_2
$[(\Lambda_k{}^t, \mathbf{U}_k^t)]_{t=t_1 \ldots t_2}$	top k eigen-pairs from time t_1 to t_2
$[\triangle(G^t)]_{t=t_1 \ldots t_2}$	$\triangle(G)$ from time t_1 to t_2
$[S(G^t)]_{t=t_1 \ldots t_2}$	$S(G)$ from time t_1 to t_2

The simplest eigen-function is the eigenvalues and eigenvectors themselves. Specifically, the eigenvalues of a graph's adjacency matrix can be used to measure the epidemic threshold or path capacity of a graph as mentioned in Chapter 3, while the eigenvectors can be used to evaluate the centrality of nodes Newman (2008), or to detect interesting subgraphs Prakash *et al.* (2010). Some more sophisticated eigen-functions include the number of triangles, network robustness (i.e., natural connectivity), eigen-gap, etc.

For all the above-mentioned network parameters (e.g., epidemic threshold, eigen centrality, number of triangles, robustness measurement, eigen-gap), it is often sufficient to use top-k eigen-pairs to achieve a high accuracy estimation of these parameters. Therefore, in order to track these parameters on a dynamic graph, we only need to track the corresponding top-k eigen-pairs at each time stamp. Formally, the eigen-function tracking problem is defined as follows. Once the top-k eigen-pairs are estimated, we can use Eq. (3.2) to (3.5) to update the corresponding eigen-functions.

Problem 1. Top-*k* Eigen-Pairs Tracking

Given: *(1) a dynamic graph G tracked from time t_1 to t_2 with starting matrix $\mathbf{A}^{t_1}$, (2) an integer k, and (3) a series of perturbation matrices $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$;*

Output: *the corresponding top-*k *eigen-pairs at each time stamp $[(\Lambda_k{}^t, \mathbf{U}_k^t)]_{t=t_1,\ldots,t_2}$.*

4.1.2 Proposed Algorithms

In this section, we present our solutions for Problem 1. We start with a baseline solution (Trip-Basic), and then present its high-order variant (Trip), followed by the attribution analysis framework for different eigen-functions and an error estimation method.

Key Idea

The key idea for Trip-Basic and Trip is to incrementally update the eigen-pairs with corresponding perturbation terms at each time stamp. By matrix perturbation theory Stewart and Sun (1990), we have the following perturbation equation

$$(\mathbf{A}^t + \Delta\mathbf{A}^t)(\mathbf{u_j}^t + \Delta\mathbf{u_j}) = (\lambda_j^t + \Delta\lambda_j)(\mathbf{u_j}^t + \Delta\mathbf{u_j}) \tag{4.2}$$

As the perturbation matrix is often very sparse, it is natural to assume that graphs in two consecutive time stamps share a fixed eigen-space. Therefore, the perturbation eigenvector $\Delta\mathbf{u_j}$ can be expressed as $\Delta\mathbf{u_j} = \sum_{i=1}^{k} \alpha_{ij}\mathbf{u_i}^t$, which is the linear combination of old eigenvectors. Taking the 2-dimensional eigen-space in Figure 4.1 as an example, the old eigenvectors are $\mathbf{u_1}^t$ and $\mathbf{u_2}^t$ marked in orange; the new eigenvectors $\mathbf{u_1}^{t+1}$ and $\mathbf{u_2}^{t+1}$ (in green) can be decomposed into old eigenvectors $\mathbf{u_1}^t$, $\mathbf{u_2}^t$ and perturbation eigenvectors $\Delta\mathbf{u_1}$, $\Delta\mathbf{u_2}$ in the same plane.

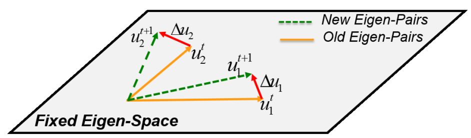


Figure 4.1: Incremental Update for Eigen-pairs Tracking.

Expanding Eq. (4.2), we get

$$\begin{aligned}&\mathbf{A}^t\mathbf{u_j}^t + \Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{A}^t\Delta\mathbf{u_j} + \Delta\mathbf{A}^t\Delta\mathbf{u_j}\\ =&\lambda_j^t\mathbf{u_j}^t + \Delta\lambda_j\mathbf{u_j}^t + \lambda_j^t\Delta\mathbf{u_j} + \Delta\lambda_j\Delta\mathbf{u_j}\end{aligned}$$

By the fact that $\mathbf{A}^t\mathbf{u_j}^t = \lambda_j^t\mathbf{u_j}^t$, the perturbation equation can be simplified as

$$\begin{aligned}&\Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{A}^t\Delta\mathbf{u_j} + \Delta\mathbf{A}^t\Delta\mathbf{u_j}\\ =&\Delta\lambda_j\mathbf{u_j}^t + \lambda_j^t\Delta\mathbf{u_j} + \Delta\lambda_j\Delta\mathbf{u_j}\end{aligned} \tag{4.3}$$

Multiplying the term $\mathbf{u_j}^{t\prime}$ on both sides; as eigenvectors are of unit length, we have

$$\mathbf{u_j}^{t\prime}\Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{u_j}^{t\prime}\Delta\mathbf{A}^t\Delta\mathbf{u_j} = \Delta\lambda_j + \mathbf{u_j}^{t\prime}\Delta\lambda_j\Delta\mathbf{u_j} \tag{4.4}$$

As we assume that $\Delta\mathbf{u_j} \ll \mathbf{u_j}$ and $\Delta\lambda_j \ll \lambda_j$, the high-order terms $\mathbf{u_j}^{t\prime}\Delta\mathbf{A}^t\Delta\mathbf{u_j}$ and $\mathbf{u_j}^{t\prime}\Delta\lambda_j\Delta\mathbf{u_j}$ in Eq. (4.4) can be discarded without losing too much accuracy.

Therefore, $\Delta\lambda_j$ can be estimated as

$$\Delta\lambda_j = \mathbf{u_j}^{t\prime}\Delta\mathbf{A}^t\mathbf{u_j}^t \tag{4.5}$$

The difference between Trip-Basic and Trip lies in their ways of estimating perturbation eigenvectors, which will be discussed below.

Trip-Basic

The Trip-Basic algorithm is a first-order eigen-pair tracking method, which ignores the high-order terms in the perturbation equation when updating eigenvectors at each time stamp. By removing the high-order terms, the perturbation equation Eq. (4.3) can be written as

$$\Delta\mathbf{A}^t\mathbf{u_j}^t + \mathbf{A}^t\Delta\mathbf{u_j} = \Delta\lambda_j\mathbf{u_j}^t + \lambda_j^t\Delta\mathbf{u_j}$$

Replacing all $\Delta\mathbf{u_j}$ terms with $\sum_{i=1}^{k}\alpha_{ij}\mathbf{u_i}^t$ and multiplying the term $\mathbf{u_p}^{t\prime}$ ($p \neq j$) on both sides, by applying the orthogonality property of eigenvectors to the new equation, we can solve the coefficient α_{pj} as

$$\alpha_{pj} = \frac{\mathbf{u_p}^{t\prime}\Delta\mathbf{A}^t\mathbf{u_j}^t}{\lambda_j^t - \lambda_p^t}$$

Therefore $\Delta\mathbf{u_j}$ can be estimated as

$$\Delta\mathbf{u_j} = \sum_{i=1, i\neq j}^{k}\left(\frac{\mathbf{u_i}^{t\prime}\Delta\mathbf{A}^t\mathbf{u_j}^t}{\lambda_j^t - \lambda_i^t}\mathbf{u_i}^t\right) \tag{4.6}$$

Suppose $\mathbf{A}^t$ is perturbed with a set of edges $\Delta E = \langle p_1, r_1\rangle, \ldots, \langle p_s, r_s\rangle$ where s is the number of non-zero elements in perturbation matrix $\Delta\mathbf{A}$. In Eq. (4.6), the term $\mathbf{u_j}^{t\prime}\Delta\mathbf{A}\mathbf{u_j}^t$ can be expanded as

$$\mathbf{u_j}^{t\prime}\Delta\mathbf{A}^t\mathbf{u_j}^t = \sum_{\langle p,r\rangle\in\Delta E}\Delta\mathbf{A}^t(p,r)\mathbf{u_{pj}}^t\mathbf{u_{rj}}^t \tag{4.7}$$

Algorithm 1 TRIP-BASIC: First-Order Eigen-Pairs Tracking

Input: Dynamic graph G tracked from time t_1 to t_2, with starting eigen-pairs $({\Lambda_k}^{t_1}, \mathbf{U}_k^{t_1})$, series of perturbation matrices $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$

Output: Corresponding eigen-pairs $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1,\ldots t_2}$

1: **for** $t = t_1$ to $t_2 - 1$ **do**
2:     **for** $j = 1$ to k **do**
3:         Initialize $\Delta\mathbf{u_j} \leftarrow \mathbf{0}$
4:         **for** $i = 1$ to $k, i \neq j$ **do**
5:             $\Delta\mathbf{u_j} \leftarrow \Delta\mathbf{u_j} + \frac{\mathbf{u_i^t}'\Delta\mathbf{A^t}\mathbf{u_j^t}}{\lambda_j^t - \lambda_i^t}\mathbf{u_i^t}$
6:         **end for**
7:         Calculate $\Delta\lambda_j \leftarrow \mathbf{u_j^t}'\Delta\mathbf{A^t}\mathbf{u_j^t}$
8:         Update $\lambda_j^{t+1} \leftarrow \lambda_j^t + \Delta\lambda_j$
9:         Update ${\mathbf{u_j}}^{t+1} \leftarrow {\mathbf{u_j}}^t + \Delta\mathbf{u_j}$
10:     **end for**
11: **end for**
12: Return $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1\ldots t_2}$

Eq. (4.6) and Eq. (4.7) naturally lead to our base solution (TRIP-BASIC) for solving Problem 1 as follows.

The approximated eigen-pairs for each time stamp is computed from step 2 to 10. Each $\Delta\lambda_j$ and $\Delta\mathbf{u}_j$ is calculated from step 3 to 7 by Eq. (4.6) and (4.7). At step 8 and 9, λ_j^t and ${\mathbf{u_j}}^t$ is updated with $\Delta\lambda_j$ and $\Delta\mathbf{u}_j$. Note that after updating the eigenvector in step 9, we normalize each of them to unit length.

Complexity Analysis. The efficiency of proposed Algorithm 1 is summarized in Lemma 1. Both time complexity and space complexity is linear w.r.t. the total number of the nodes in the graph (n) and total number of the time stamps (T).

Lemma 1. Complexity of First-Order Eigen-Function Tracking. *Suppose T is the total number of the time stamps, s is the average number of perturbed edges in $[\Delta \mathbf{A}^t]_{t=t_1,\ldots t_2-1}$, then the time cost for Algorithm 1 is $O(Tk^2(s+n))$; the space cost is $O(Tnk+s)$.*

Proof. In each time stamp from time t_1 to t_2-1, top k eigen-pairs are updated in steps 2-10. By Eq. (4.7), the complexity of computing term $\mathbf{u_j}^{t\prime} \Delta \mathbf{A}^t \mathbf{u_j}^t$ is $O(s)$, so the overall complexity of step 5 is $O(s+n)$. Therefore calculating $\Delta \mathbf{u_j}$ from step 4 to 6 takes $O(k(s+n))$. In step 7, computing $\Delta\lambda_j$ takes another $O(s)$. Updating λ_j^t and $\mathbf{u_j}^t$ in step 8 and 9 takes $O(1)$ and $O(n)$. Therefore updating all top-k eigen-pairs $\mathbf{U_k}^t$ and Λ_k^t takes $O(k^2(s+n))$ and $O(ks)$ respectively. Thus the overall time complexity for T iterations is $O(Tk^2(s+n))$.

For space cost, it takes $O(k)$ and $O(nk)$ to store Λ_k^t and $\mathbf{U_k}^t$ at each time stamp. In the update phase from step 2 to 10, it takes $O(s)$ to store $\Delta \mathbf{A}^t$, $O(1)$ to update λ_j^t and $O(n)$ to update $\mathbf{u_j}^t$. However, the space used in the update phase can be reused in each iteration. Therefore the overall space complexity for T time stamps takes a space of $O(Tnk+s)$. □

Trip

The baseline solution in Algorithm 1 is simple and straight-forward, but it has the following limitations. First, the approximation error of first-order matrix perturbation is in the order of $\|\Delta \mathbf{A}^t\|$. In other words, the quality of such approximation might decrease quickly w.r.t. the increase of $\|\Delta \mathbf{A}^t\|$. Second, the approximation quality is highly sensitive to the small eigen-gap of $\mathbf{A}^t$ as indicated by Eq. (4.6). In order to address these limitations, we further propose Algorithm 2 by adopting the high-order matrix perturbation to update the eigen-pairs of $\mathbf{A}^{t+1}$. The main difference between Algorithm 2 and Algorithm 1 is that we take high-order terms in the perturbation equation (Eq. (4.3)) into consideration while updating eigenvectors. Similar to TRIP-BASIC we replace all $\Delta\mathbf{u_j}$ terms with $\sum_{i=1}^{k}\alpha_{ij}\mathbf{u_i}^t$ and multiplying the term $\mathbf{u_p}^{t\prime}$ (for $1 \leq p \leq k$, $p \neq j$) on both sides. By applying the orthogonality property of eigenvectors to the new equation, we have

$$\mathbf{X}^t(p,j) + \alpha_{pj}\lambda_p^t + \sum_{i=1}^{k}\mathbf{X}^t(p,i)\alpha_{ij} = \alpha_{pj}\lambda_j^t + \alpha_{pj}\Delta\lambda_j$$

where $\mathbf{X}^t = \mathbf{U}_k^{t\prime}\Delta\mathbf{A}^t\mathbf{U}_k^t$. Reorganizing the terms in the above equation, we have

$$\mathbf{X}^t(p,j) - \alpha_{pj}(\lambda_j^t + \Delta\lambda_j - \lambda_p^t) + \sum_{i=1}^{k}\mathbf{X}^t(p,i)\alpha_{ij} = 0$$

By defining $\mathbf{v} = \lambda_j^t + \Delta\lambda_j - \lambda_p^t$ for $p = 1, \ldots, k$, $\mathbf{D}^t = diag(\mathbf{v})$ and $\alpha_{\mathbf{j}} = [\alpha_{1j}, \ldots, \alpha_{kj}]$, the above equation can be expressed as

$$\mathbf{X}^t(:,j) - \mathbf{D}^t\alpha_{\mathbf{j}} + \mathbf{X}^t\alpha_{\mathbf{j}} = \mathbf{0}$$

Solve the above equation for $\alpha_{\mathbf{j}}$, we have

$$\alpha_{\mathbf{j}} = (\mathbf{D}^t - \mathbf{X}^t)^{-1}\mathbf{X}^t(:,j)$$

In Algorithm 2, the top-k eigen-pairs at each time stamp is updated from step 2 to 11. In step 2, matrix $\mathbf{X}^t$ is calculated for computing $\Delta\Lambda_k$ and $\Delta\mathbf{U}_k$. In step 4, all top-k eigenvalues Λ_k are updated by $\Delta\Lambda_k$. From step 6 to 10, each $\mathbf{u_j}^t$ is updated according to the derivations of the eigen-update rule in mentioned above. Again, after we update the eigenvector in step 9, we normalize each of them to unit length.

Complexity Analysis. The efficiency of Algorithm 2 is given in Lemma 2. Compared with TRIP-BASIC, both time and space complexity are still linear w.r.t. total

[1] Here the *diag* function works the same with the one in Matlab. When applying to a matrix, *diag* returns a vector of the main diagonal elements of the matrix; when applying to a vector, it returns a square diagonal matrix with the elements of the vector on the main diagonal.

Algorithm 2 TRIP: High-Order Eigen-Pairs Tracking

Input: Dynamic graph G tracked from time t_1 to t_2, with starting eigen-pairs $({\Lambda_k}^{t_1}, \mathbf{U}_k^{t_1})$, series of perturbation matrices $[\Delta \mathbf{A}^t]_{t=t_1,\dots t_2-1}$

Output: Corresponding eigen-pairs $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1,\dots t_2}$

1: **for** $t = t_1$ to $t_2 - 1$ **do**

2:     Calculate $\mathbf{X}^t \leftarrow \mathbf{U}_k^{t\,\prime} \Delta \mathbf{A}^t \mathbf{U}_k^t$

3:     $\Delta \Lambda_k \leftarrow diag(X^t)$[1]

4:     Update $\Lambda_k^{t+1} \leftarrow \Lambda_k^t + \Delta \Lambda_k$

5:     **for** $j = 1$ to k **do**

6:         Calculate $\mathbf{v} \leftarrow \lambda_j^t + \Delta\lambda_j - \lambda_p^t$ for $p = 1, \dots, k$

7:         $\mathbf{D}^t \leftarrow diag(\mathbf{v})$

8:         Calculate $\alpha_{\mathbf{j}} \leftarrow (\mathbf{D}^t - \mathbf{X}^t)^{-1} \mathbf{X}(:, j)$

9:         Calculate $\Delta \mathbf{u_j} \leftarrow \sum_{i=1}^{k} \alpha_{ij} {\mathbf{u_i}}^t$

10:         Update ${\mathbf{u_j}}^{t+1} \leftarrow {\mathbf{u_j}}^t + \Delta \mathbf{u_j}$

11:     **end for**

12: **end for**

13: Return $[({\Lambda_k}^t, \mathbf{U}_k^t)]_{t=t_1+1\dots t_2}$

number of nodes in the graph and the total number of time stamps, with a slight increase in k, which is often very small.

Lemma 2. Complexity of High-Order Eigen-Function Tracking. *Suppose T is the total number of time stamps, s is the average number of perturbed edges in $[\Delta \mathbf{A}^\mathbf{t}]_{t=t_1,\dots t_2-1}$, then the time cost for Algorithm 2 is $O(T(k^4 + k^2(n+s)))$; the space cost is $O(Tnk + k^2 + s)$.*

Proof. In each time stamp from time t_1 to $t_2 - 1$, top k eigen-pairs are updated in steps 2-11. Using the update rule provided in Eq. (4.7), calculating $\mathbf{X}^t$ in step 2 takes

$O(k^2 s)$. Updating top eigenvalues in step 3-4 takes $O(k)$. From step 5-11, eigenvectors are updated. It takes $O(k^3)$ in to do matrix inversion and multiplication in step 8 and $O(nk)$ to calculate $\Delta\mathbf{u_j}$ in step 9. Therefore updating $\mathbf{U}_k^t$ takes $O(k^4 + nk^2))$. Thus the overall time complexity for T iterations takes $O(T(k^4 + k^2(n + s)))$.

For space cost, it takes $O(k)$ and $O(nk)$ to store Λ_k^t and $\mathbf{U_k}^t$, $O(s)$ to store $\Delta\mathbf{A}^t$ for each time stamp. In the update phase from step 2 to 11, it takes $O(k^2)$ to store and calculate $\mathbf{X}^t$, $\mathbf{D}^t$; $O(k)$ to store v and $\alpha_{\mathbf{j}}$; $O(k^2)$ to calculate $\alpha_{\mathbf{j}}$. However the space cost in the update phase can be reused in each iteration. Therefore the overall space complexity for T time stamps takes a space of $O(Tnk + k^2 + s)$. □

Attribution Analysis

Based on our TRIP algorithms, we can effectively track the corresponding eigen-functions of interest. In reality, we might also be interested in understanding the key factors that cause these changes in dynamic graphs. For example, among all the changed edges in $\Delta\mathbf{A}$, which edge is most important in causing the increase/decrease of the epidemic threshold, or the number of triangles, etc. The importance of an edge $\langle p, r\rangle \in \Delta E$ can be measured as the change it can make on the corresponding eigen-functions, which can be written as

$$score(\langle p, r\rangle) \sim \Delta g_{\langle p,r\rangle} = g_{G\cup\langle p,r\rangle} - g_G$$

where $g(.)$ is one of the eigen-functions we define Section 4.1.1.

In Algorithm 3, all removed edges and added edges are extracted from $\Delta\mathbf{A}$ in step 1 and step 2. The impact score of each removed edge at time t is calculated from step 3 to 5. Similarly, the score of each added edge is calculated from step 6 to 8. In the end, top l removed edges and l added edges are returned as high impact edges at time t.

Algorithm 3 Dynamic Attribution Analysis

Input: Dynamic graph G and eigen-pairs $({\Lambda_k}^t, \mathbf{U}_k^t)$ at time t, perturbation matrix $\Delta\mathbf{A}^t$, eigen-function $g(.)$, number l

Output: top l added edges and removed edges at time t that have the largest impact on eigen-function $g(.)$

1: *removed* $\leftarrow$ extract all removed edges in $\Delta\mathbf{A}^t$
2: *added* $\leftarrow$ extract all added edges in $\Delta\mathbf{A}^t$
3: **for** each edge $\langle p, r\rangle$ in *removed* **do**
4: $\quad$ $score(\langle p, r\rangle) \leftarrow g_{G^t} - g_{G^t \setminus \langle p,r\rangle}$
5: **end for**
6: **for** each edge $\langle p, r\rangle$ in *added* **do**
7: $\quad$ $score(\langle p, r\rangle) \leftarrow g_{G^t \cup \langle p,r\rangle} - g_{G^t}$
8: **end for**
9: Return top l edges in *removed* and *added* with highest scores respectively

Complexity Analysis. Assume that the complexity of calculating $\Delta g_{\langle p,r\rangle}$ is $h(n, k, s)$, where h is a function of number of nodes n, number of eigen-pairs k and number of changed edges s. Then the complexity of calculating the impact scores of all changed edges (from step 3 to 8) is $O(sh(n, k, s))$. Given the impact score of each changed edges, the complexity of picking out top l edges from *removed* and *added* set using heap structure is $O(|removed|\text{log}l) + O(|added|\text{log}l) = O(s\text{log}l)$. Therefore the overall complexity for attribution analysis at time t is $O(s(h(n, k, s) + \text{log}l))$.

Error Estimation

The core mechanism for both Trip-Basic and Trip is to incrementally update the eigen-pairs at each time stamp. With this scheme, the tracking error of eigen-pairs would accumulate as time goes by. Therefore, finding a proper time to restart the algorithm is of key importance to keep the tracking error within a reasonable range. For simplicity, we only estimate the error of leading eigenvalue since it is the key part for most of the eigen-functions. Here we denote $err(\lambda^t)$ as the estimated error on λ introduced at time t. Intuitively, $err(\lambda^t)$ would be strongly correlated to the impact of $\Delta\mathbf{A}^t$ on the original eigen-space. As the original eigen-space is defined by the top-k eigenvectors $\mathbf{U}_k^{t_1}$ at the first time stamp t_1, to measure the impact of $\Delta\mathbf{A}^t$ on $\mathbf{U}_k^{t_1}$, we can project $\Delta\mathbf{A}^t$ into this space and take the Frobenius norm of the projection as its actual impact. Eq. (4.8) formalizes the impact function of $\Delta\mathbf{A}^t$ on eigen-space $\mathbf{U}_k^{t_1}$.

$$err(\lambda^t) \sim impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1}) = \| \mathbf{U}_k^{t_1}\mathbf{U}_k^{t_1\prime}\Delta\mathbf{A}^t \|_{Fro} \tag{4.8}$$

We denote the summation of the perturbation impacts from the first time stamp t_1 to current stamp t as $err_{acc}(\lambda^t)$. This number can be viewed as a good approximation of accumulated tracking error on leading eigenvalue from t_1 to t. In other words, the curve of $err_{acc}(\lambda^t)$ from $t = t_1, \ldots, t_2$ would have a similar shape with real tracking error curve of TRIP algorithms.

Algorithm 4 Error Estimation for Eigen-function Tracking

Input: Dynamic graph G tracked from time t_1 to t_2, with starting eigen-pairs $({\Lambda_k}^{t_1}, \mathbf{U}_k^{t_1})$, series of perturbation matrices $[\Delta\mathbf{A}^t]_{t=t_1,\ldots t_2-1}$

Output: Corresponding estimated error $err_{acc}(\lambda^t)$ for $t = t_1+1, \ldots t_2$

1: Initialize $err_{acc}(\lambda^{t_1}) \leftarrow 0$

2: Initialize $P \leftarrow \mathbf{U}_k^{t_1}\mathbf{U}_k^{t_1\prime}$

3: **for** $t = t_1 + 1$ to t_2 **do**

4: Calculate $impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1}) \leftarrow \| P\Delta\mathbf{A}^t \|_{Fro}$

5: Calculate $err_{acc}(\lambda^t) \leftarrow err_{acc}(\lambda^{t-1}) + impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1})$

6: **end for**

7: Return $err_{acc}(\lambda^t)$ for $t = t_1 + 1, \ldots t_2$

In Algorithm 4, $err_{acc}(\lambda^{t_1})$ is initialized as 0 in step 1 and P is initialized as the projection matrix in step 2. From step 3 to 6, the impact of each perturbation matrix is calculated and accumulated to $err_{acc}(\lambda^t)$. In step 7, the estimated error array $err_{acc}(\lambda^t)$ for $t = t_1 + 1, \ldots t_2$ is returned.

Complexity Analysis. The complexity of initializing projection matrix P is $O(n^2k)$. Since $\Delta\mathbf{A}^t$ is often very sparse, the complexity of calculating $impact(\Delta\mathbf{A}^t, \mathbf{U}_k^{t_1})$ can be reduced to $O(ns)$ where s is the number of changed edges at current time stamp. The complexity of accumulating $err_{acc}(\lambda^t)$ at each time stamp is $O(1)$. Therefore the overall time complexity for error estimation over time series of length T is $O(n^2k + Tns)$.

4.1.3 Experimental Evaluation

In this section, we evaluate Trip-Basic and Trip on real datasets. All the experiments are designed to answer the following two questions

- *Effectiveness*: how accurate are our algorithms in tracking eigen-functions, analyzing corresponding attributions and estimating the tracking errors?
- *Efficiency*: how fast are the tracking algorithms?

Experiment Setup

Machine. We ran our experiment in a machine with 2 processors Intel Xeon 3.5GHz with 256GB of RAM. Our experiment is implemented with Matlab using single thread.

Datasets. Here we use three real datasets for evaluations.

AS The first dataset we use for the evaluation is Autonomous system graph, which is available at `http://snap.stanford.edu/data/`. The graph has recorded communications between routers in the Internet for a long period of time. Based on the data from `http://www.routeviews.org`, we constructed an undirected dynamic communication graph that contains 100 daily instances with time span from Nov 8 1997 to Feb 16 1998. The largest graph among those instances has 3,569 nodes and 12,510 edges. The dataset shows both the addition and deletion of nodes and edges over time.

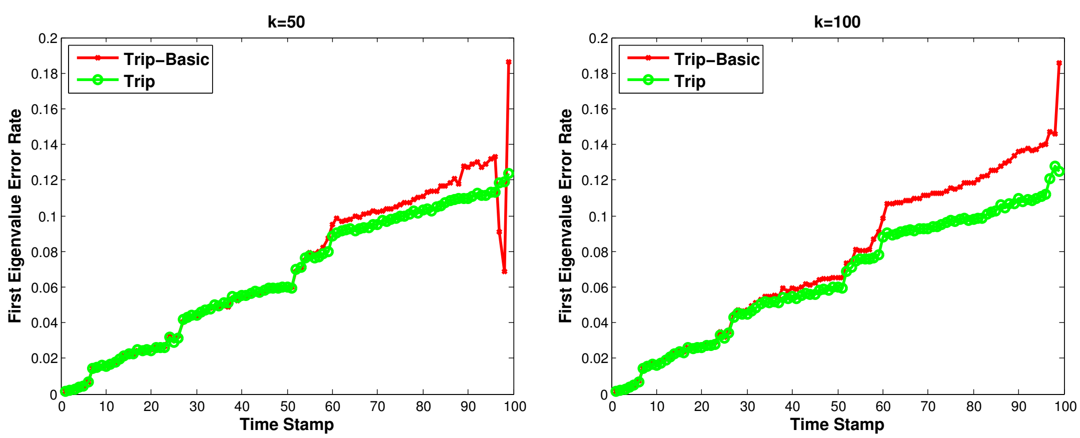
Power Grid The second dataset is power grid network. It is a static, undirected, un-weighted network representing the topology of the Western States Power Grid of the United State Watts and Strogatz (1998), which has 4,941 nodes and 6,594 edges. To simulate the evolving process, we randomly add $0.5\%m$ (m is the number of edges in the graph) new edges to the graph at each time stamp as perturbation edges. We have changed different percentages of perturbation edges, and experimented several runs on each of the settings. As the results are similar, we only report the results from one run for brevity.

Airport The third dataset is a static, undirected, un-weighted airport network, which represents the internal US air traffic lines between 2,649 airports and has 13,106 links (available at `http://www.levmuchnik.net/Content/Networks/NetworkData.html`). Again, similar synthetic evolving process was done on this dataset. With similar experiment results, we only report those from one run of simulation for brevity.

Evaluation Metrics. For the quality of eigen-functions tracking, we use the error rate ϵ. For eigenvalues, number of triangles and robustness measurement, their error rate are computed as $\epsilon = \frac{|g-g^*|}{g^*}$, where g and g^* are the estimated and true eigen-function values, respectively. For eigenvector, the error is computed as $\epsilon = 1 - \frac{\mathbf{u}\mathbf{u}^*}{\|\mathbf{u}\|\|\mathbf{u}^*\|}$, where $\mathbf{u}$ is the estimated eigenvector and $\mathbf{u}^*$ is the corresponding true eigenvector. For attribution analysis, we use the top-10 precision. For efficiency, we report the speedup of our algorithms over the re-computing strategy which computes the corresponding eigen-pairs from scratch at each time stamp.

Effectiveness Results

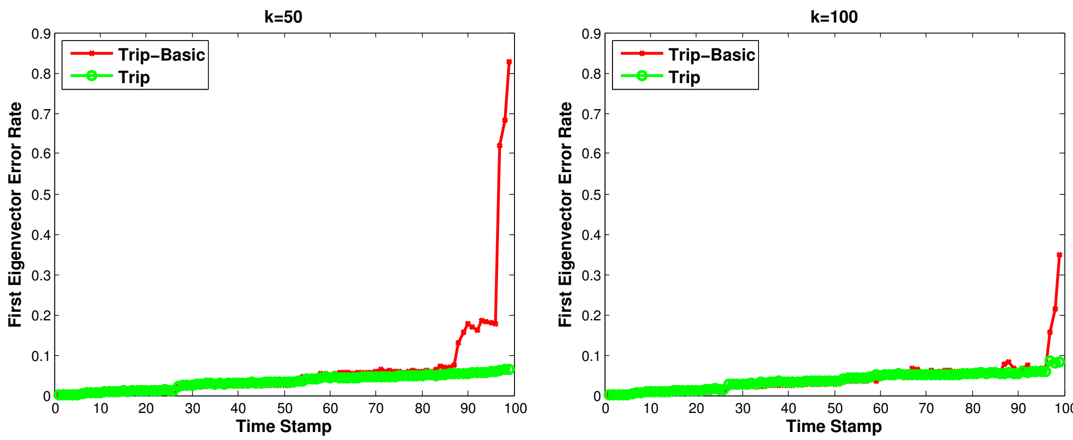
Effectiveness of Eigen-Function Tracking. Figure 4.2 to 4.6 compare the effectiveness of Trip-Basic and Trip using different number of eigen-pairs (k). We have the following observations. First, for all of the four eigen-functions, both algorithms could reach an overall error rate below 20% at the end of the tracking process. Second, when k is increased from 50 to 100, Trip-Basic could get a relatively more stable approximation over the tracking process. Third, Trip is more stable and overall reaches a smaller error rate compared with Trip-Basic. For example, as time goes by, Trip-Basic starts to fluctuate sharply when $k = 50$ on all four eigen-functions. Finally, the error on the number of triangles is relatively higher. This is probably because the number of triangles is the sum of cubic eigenvalues, and small errors on eigenvalues would therefore be magnified on the final result.



(a) k=50 (b) k=100

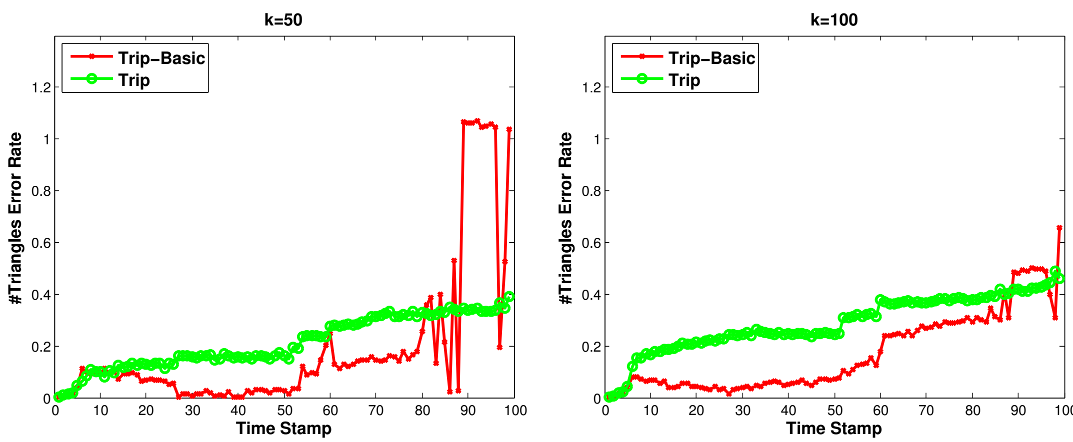
Figure 4.2: The Error Rate of First Eigenvalue Approximation.

In addition, we also compared our algorithms with three different eigen-pair estimation methods, which include (1) "QR Decom", a QR decomposition based eigen-pairs updating method Li *et al.* (2015b); (2) "SVD delta", simple SVD decomposition on $\Delta\mathbf{A}$; and (3) "Nystrom", a sampling-based eigen-pair estimation method



(a) k=50 (b) k=100

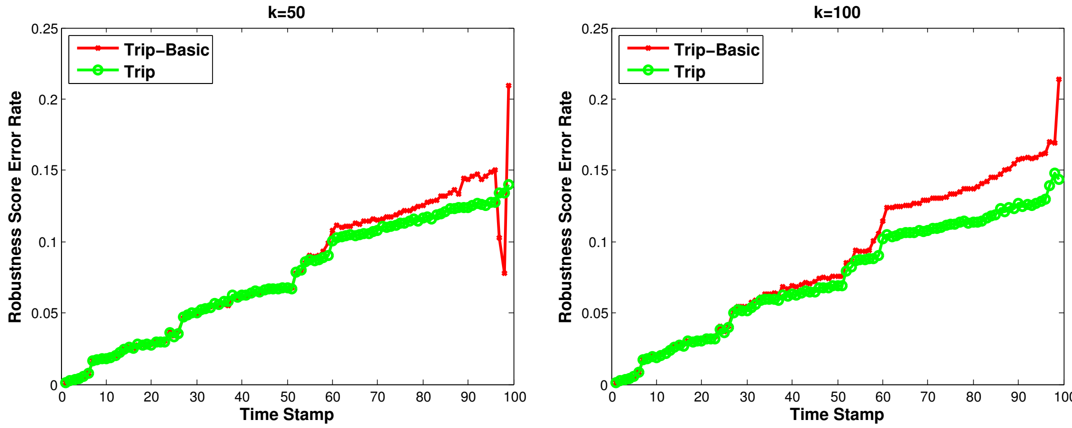
Figure 4.3: The Error Rate of First Eigenvector Approximation.



(a) k=50 (b) k=100

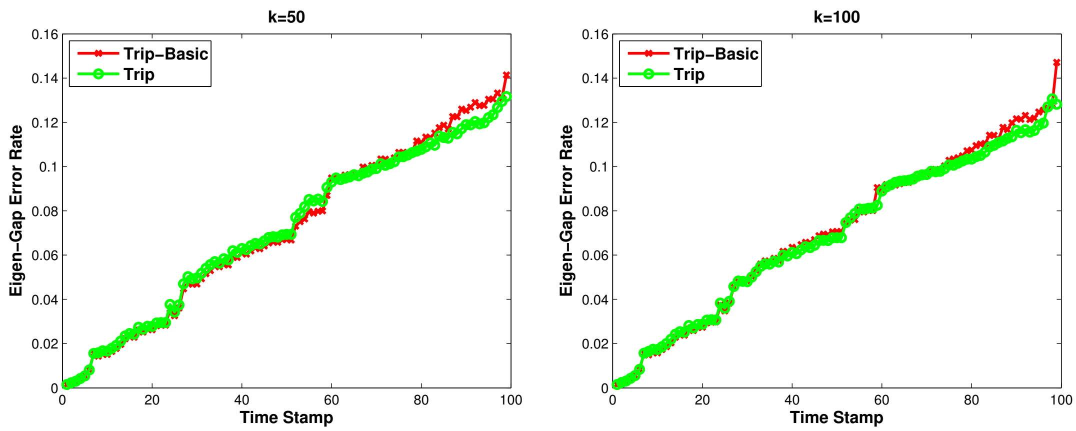
Figure 4.4: The Error Rate of Number of Triangles Approximation.

derived from Nystrom algorithm Drineas and Mahoney (2005). For better effectiveness/efficiency trade-off, we sample 2,000 nodes for Nystrom algorithm to calculate eigen-pairs in our experiment. To better illustrate the results, we take the error rates of all methods for every 15 days on the AS data set. As “SVD delta” method causes large tracking errors compared to other methods, we only report the error rates from other comparing methods as shown from Figure 4.7 to Figure 4.11. We can see that



(a) k=50 (b) k=100

Figure 4.5: The Error Rate of Robustness Score Approximation.

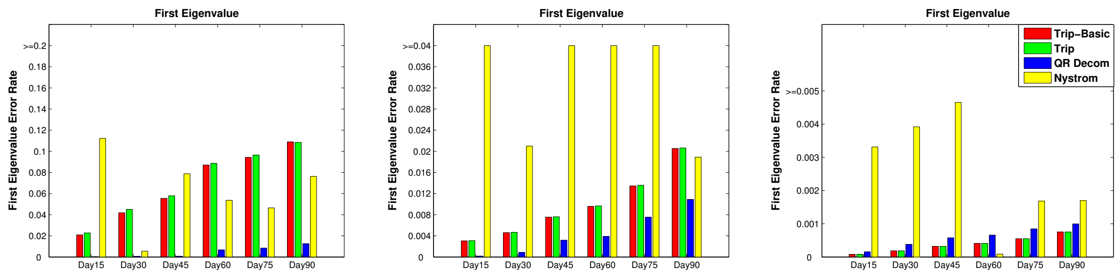


(a) k=50 (b) k=100

Figure 4.6: The Error Rate of Eigen-gap Approximation.

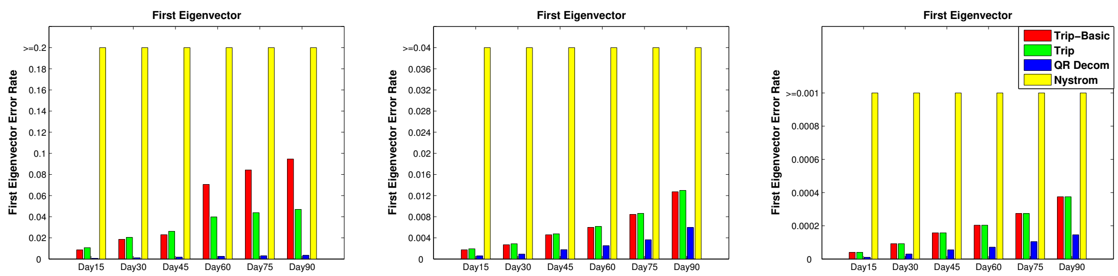
the performance of Trip-Basic and Trip are among the best methods though their error rates keep increasing as time accumulates.

Effectiveness of Attribution Analysis. For attribution analysis, we divided the changed edges at each time stamp into two classes: edges being added and edges being removed. And among these two classes, we rank those edges according to their attribution score defined in Section 4.1.2. As a consequence, the top-ranked edges are the ones that have largest impact on the corresponding eigen-functions. Here we



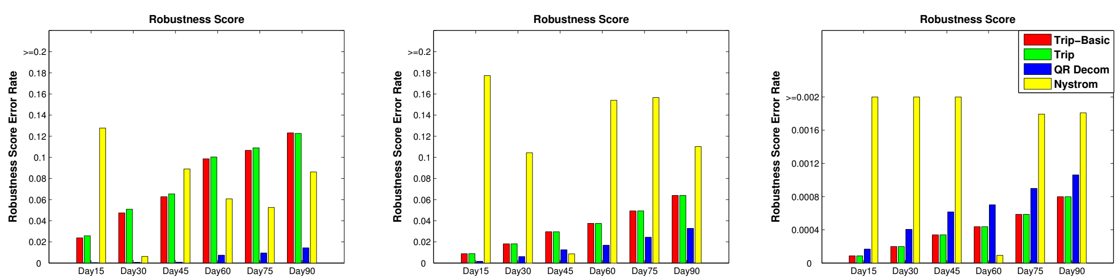
(a) AS (b) Power Grid (c) Airport

Figure 4.7: The Error Rate of First Eigenvalue Approximation.



(a) AS (b) Power Grid (c) Airport

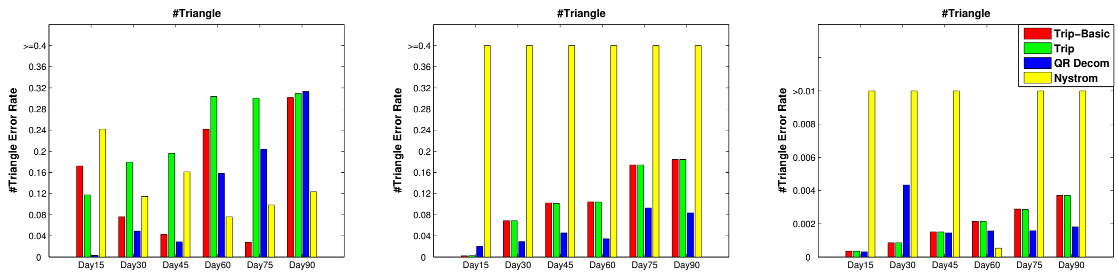
Figure 4.8: The Error Rate of First Eigenvector Approximation.



(a) AS (b) Power Grid (c) Airport

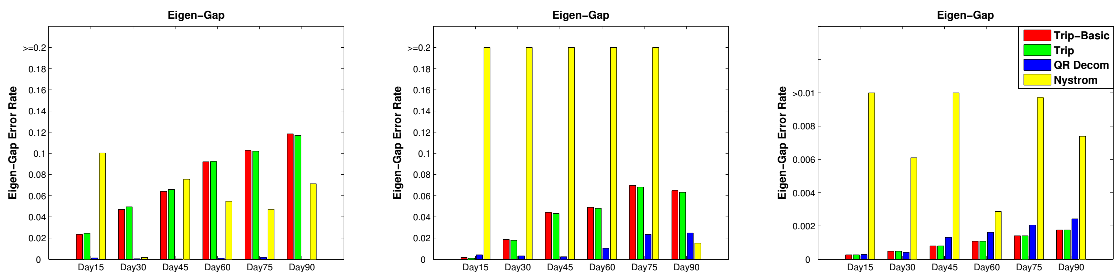
Figure 4.9: The Error Rate of Robustness Score Approximation.

scored and ranked those edges with our approximated eigen-pairs and true eigen-pairs respectively and then compare the similarity between the two ranks. The precision of attribution analysis therefore is defined as the precision at rank 10 in approximated



(a) AS (b) Power Grid (c) Airport

Figure 4.10: The Error Rate of Number of Triangles Approximation.

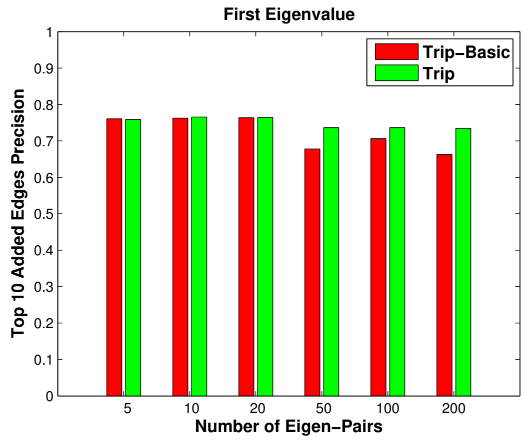


(a) AS (b) Power Grid (c) Airport

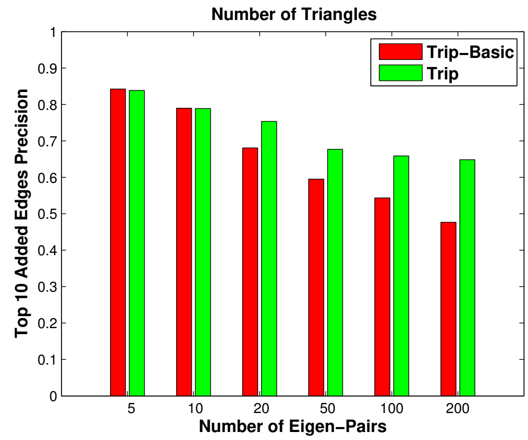
Figure 4.11: The Error Rate of Eigen-gap Approximation.

rank list. As similar results are observed in all three data set, we only report those on AS dataset as shown in Figure 4.12 and 4.13. For the analysis on both added edges and removed edges, TRIP overall outperforms TRIP-BASIC.

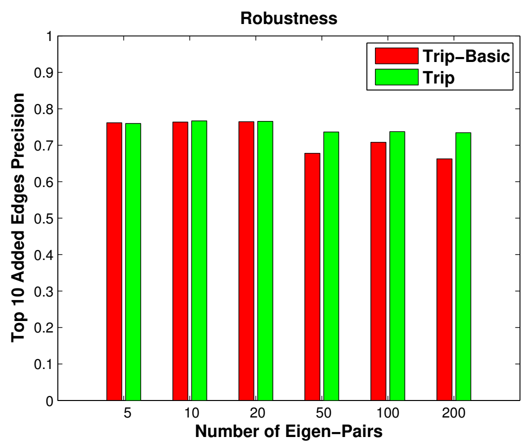
Effectiveness of Error Estimation. To show the effectiveness of Algorithm 4, we compare the curve shapes between true errors of TRIP and accumulative estimated errors $err_{acc}(\lambda^t)$ on AS data set with $k = 50$. Ideally, the two curves should overlap with each other when $err_{acc}(\lambda^t)$ is properly scaled with some elaborately picked factor. Figure 4.14 shows that the estimated error $err_{acc}(\lambda^t)$ can effectively catch sharp error increases in the tracking process as marked in red circle. Therefore, it can be used as a trigger to restart the tracking process so that the accumulative error can always be



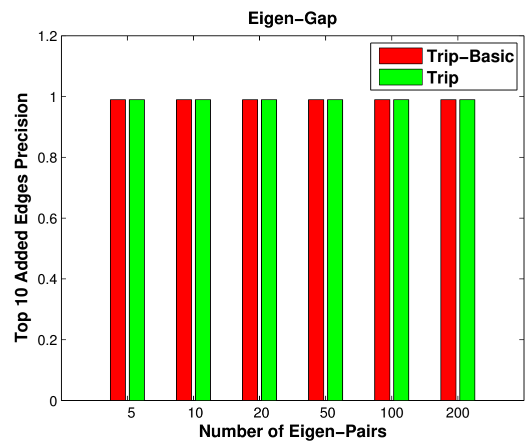
(a) First Eigenvalue



(b) Number of Triangles



(c) Robustness



(d) Eigen-gap

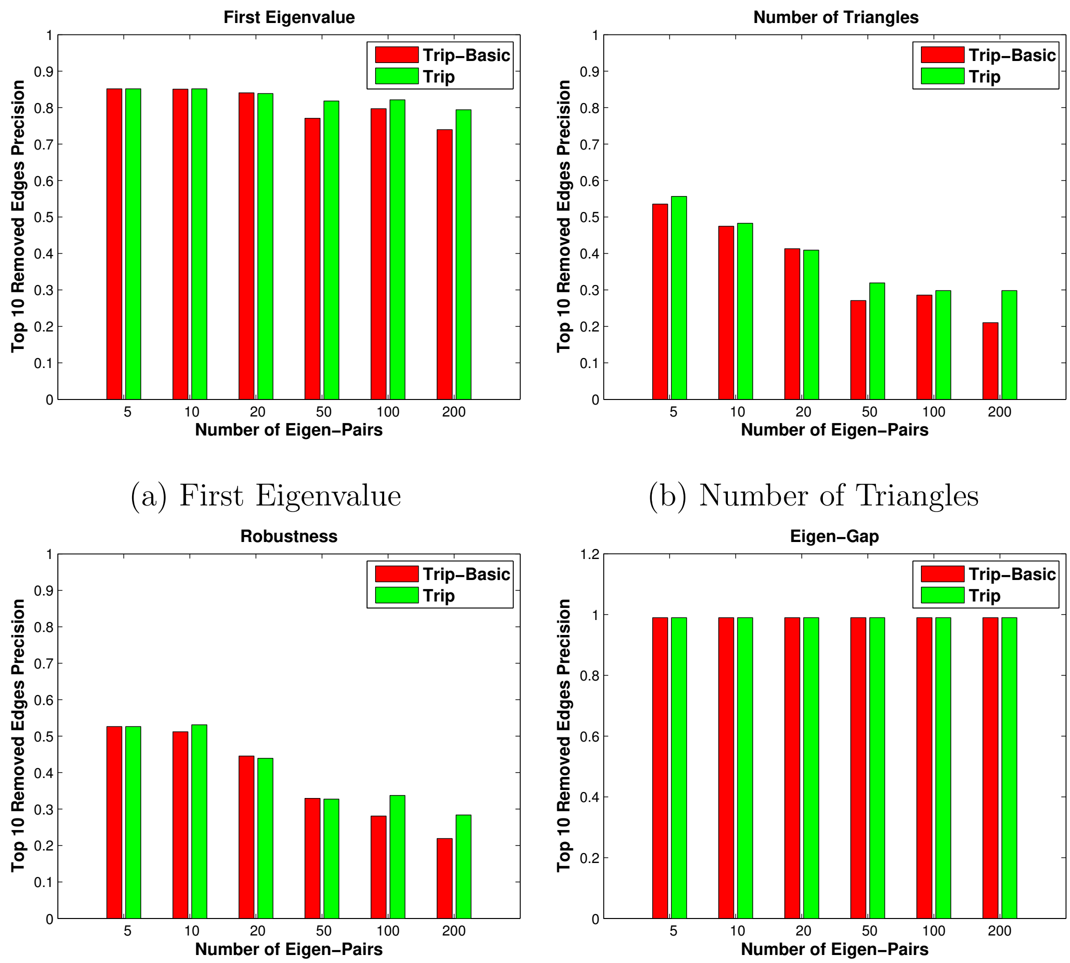
Figure 4.12: Average Precision over Time for the Attribution Analysis (Added Edges).

kept within a low range.

Efficiency Results

Figure 4.15 shows the average speed up w.r.t. different k values on AS dataset. We see that both TRIP-BASIC and TRIP can achieve more than 20$\times$ speed up when k is small. As k increases, the speedup decreases.

To further demonstrate the efficiency of the proposed algorithms, we also com-



(a) First Eigenvalue (b) Number of Triangles

(c) Robustness (d) Eigen-gap

Figure 4.13: Average Precision over Time of the Attribution Analysis (Removed Edges).

pare their effectiveness/efficiency trade-offs with those of the alternative methods. Figure 4.16 shows that our algorithms can keep the average error rate very small on all three data sets while consuming least amount of time.

4.2 Cross-layer Dependency Inference

The interactions observed across different domains have facilitated the emergence of multi-layered network. Examples of such complicated network systems include

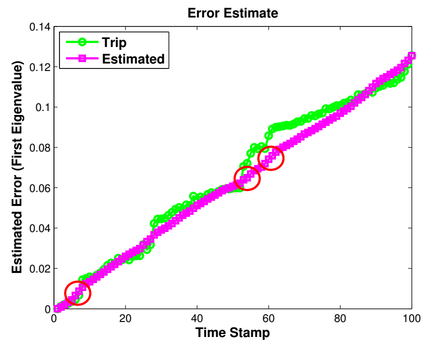


Figure 4.14: The Estimated Error of TRIP-BASIC and TRIP on **AS** Data Set.

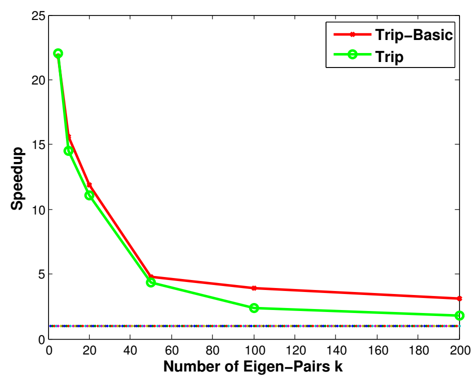
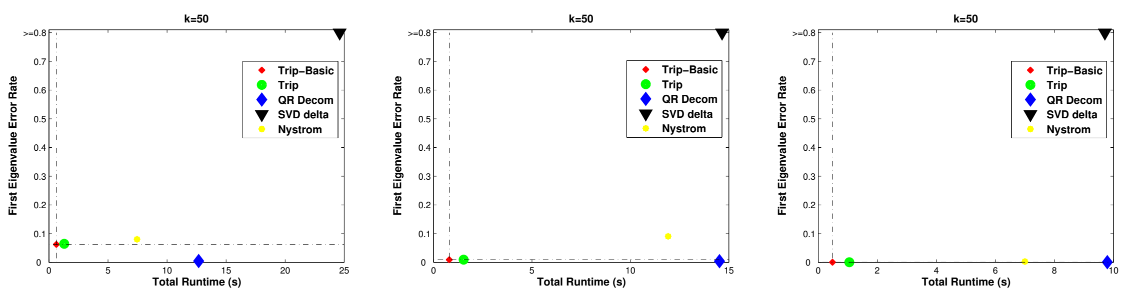


Figure 4.15: The Running Time Speedup of TRIP-BASIC and TRIP W.R.T. to k.



(a) AS (b) Power Grid (c) Airport

Figure 4.16: The Error Rate Vs. Total Runtime of First Eigenvalue Approximation in 100 Time Stamps.

critical infrastructure network mentioned in Chapter 1, collaboration platforms, and biological systems. One crucial topological structure that differentiates multi-layered network to other network models is its *cross-layer dependency*, which describes the associations/dependencies between the nodes from different layers. For example, in infrastructure networks, the full functioning of the AS network depends on the sufficient power supply from the power grid layer, which in turn relies on the functioning of the transportation network (e.g., to deliver the sufficient fuel). Similarly, in the biological systems, the dependency is represented as the associations among diseases, genes, and drugs. In practice, the cross-layer dependency often plays a central role in many multi-layered network mining tasks. For example, in the critical infrastructure network, the intertwined cross-layer dependency is considered as a major factor of system robustness. This is because a small failure on the supporting network (e.g., power station malfunction in power grid) may be amplified in all its dependent networks through cross-layer dependencies, resulting in a catastrophic/cascading failure of the entire system. On the other hand, the cross-layer dependency in the biological system is often the key to new discoveries, such as new treatment associations between existing drugs and new diseases.

In spite of its key importance, it remains a daunting task to know the exact cross-layer dependency structure in a multi-layered network, due to noise, incomplete data sources, limited accessibility to network dynamics. For example, an extreme weather event might significantly disrupt the power grid, the transportation network and the cross-layer dependencies in between at the epicenter. Yet, due to limited accessibility to the damaged area during or soon after the disruption, the cross-layer dependency structure might only have a probabilistic and/or coarse-grained description. On the other hand, for a newly identified chemical in the biological system, its cross-layer dependencies w.r.t. proteins and/or the diseases might be completely unknown due to clinical limitations. (i.e., the *zero-start* problem).

In this work, we aim to tackle the above challenges by developing effective and efficient methods to infer cross-layer dependency on multi-layered networks. The main contributions of the work can be summarized as

- *Problem Formulations:* We define the cross-layer dependency inference problem as a regularized optimization problem. The key idea behind this formulation is to collectively leverage the within-layer topology and the observed cross-layer dependency to infer a latent, low-rank representation for each layer, which can be used to infer the missing cross-layer dependencies in the network.

- *Algorithms and Analysis:* We propose an effective algorithm—FASCINATE to infer the cross-layer dependency on multi-layered networks, and analyze its optimality, convergence and complexity. We further present its variants and generalizations, including an online algorithm to address the *zero-start* problem.

- *Evaluations:* We perform extensive experiments on five real datasets to substantiate the effectiveness, efficiency, and scalability of our proposed algorithms. Specifically, our experimental evaluations show that the proposed algorithms outperform their best competitors by 8.2%-41.9% in terms of inference accuracy while enjoying linear complexity. Moreover, the proposed FASCINATE-ZERO algorithm can achieve up to $10^7\times$ speedup with barely any compromise on accuracy.

4.2.1 Problem Definition

In this section, we give the formal definitions of the cross-layer dependency inference problems. The main symbols used in this work are listed in Table 5.3. Following the convention, we use bold upper-case for matrices (e.g., $\mathbf{A}$), bold lower-case for vectors (e.g., $\mathbf{a}$) and calligraphic for sets (e.g., $\mathcal{A}$). $\mathbf{A}'$ denotes the transpose of matrix $\mathbf{A}$. We use the $\hat{}$ sign to denote the notations after a new node is accommodated to the system (e.g., $\hat{J}$, $\hat{\mathbf{A}}_1$), and the ones without the $\hat{}$ sign as the notations before the new node arrives.

Table 4.2: Main Symbols FASCINATE.

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}$	the adjacency matrices (bold upper case)
$\mathbf{a}, \mathbf{b}$	column vectors (bold lower case)
$\mathcal{A}, \mathcal{B}$	sets (calligraphic)
$\mathbf{A}(i, j)$	the element at i^{th} row j^{th} column in 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 $\mathbf{A}$
$\hat{\mathbf{A}}$	the adjacency matrix of $\mathbf{A}$ with the newly added node
$\mathbf{G}$	the layer-layer dependency matrix
$\mathcal{A}$	within-layer connectivity matrices of the network $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_g\}$
$\mathcal{D}$	cross-layer dependency matrices $\mathcal{D} = \{\mathbf{D}_{i,j}\ i, j = 1, ..., g\}$
$\mathbf{W}_{i,j}$	weight matrix for $\mathbf{D}_{i,j}$
$\mathbf{F}_i$	low-rank representation for layer-i ($i = 1, ..., g$)
m_i, n_i	number of edges and nodes in graph $\mathbf{A}_i$
$m_{i,j}$	number of dependencies in $\mathbf{D}_{i,j}$
g	total number of layers
r	the rank for $\{\mathbf{F}_i\}_{i=1,...,g}$
t	the maximal iteration number
ξ	the threshold to determine the iteration

While several multi-layered network models exist in the literature (See Chapter 2 for a review), we will focus on a recent model proposed in Chen *et al.* (2015), due to its flexibility to model more complicated cross-layer dependency structure. We refer the readers to Chapter 3 for its full details. For the purpose of this work, we mainly need the following notations to describe a multi-layered network with g layers. First, we need a $g \times g$ layer-layer dependency matrix $\mathbf{G}$, where $\mathbf{G}(i, j) = 1$ if layer-j depends on layer-i, and $\mathbf{G}(i, j) = 0$ otherwise. Second, we need a set of g within-layer

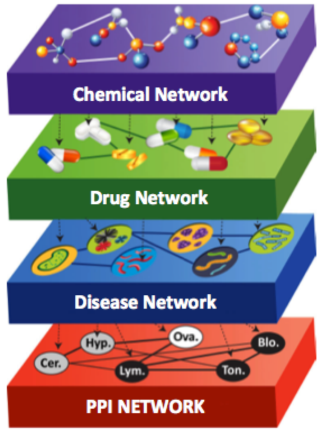


Figure 4.17: A Simplified 4-layered Network for Biological Systems.

connectivity matrices: $\mathcal{A} = \{\mathbf{A}_1, ..., \mathbf{A}_g\}$ to describe the connectivities/similarities between nodes within the same layer. Third, we need a set of cross-layer dependency matrices $\mathcal{D} = \{\mathbf{D}_{i,j}\ i, j = 1, ..., g\}$, where $\mathbf{D}_{i,j}$ describes the dependencies between the nodes from layer-i and the nodes from layer-j if these two layers are directly dependent (i.e., $\mathbf{G}(i, j) = 1$). When there are no direct dependencies between the two layers (i.e., $\mathbf{G}(i, j) = 0$), the corresponding dependency matrix $\mathbf{D}_{i,j}$ is absent. Taking the multi-layered network in Figure 4.17 for an example, the abstract layer-layer dependency network $\mathbf{G}$ of this biological system can be viewed as a line graph. The four within-layer similarity matrices in $\mathcal{A}$ are the *chemical network* ($\mathbf{A}_1$), the *drug network* ($\mathbf{A}_2$), the *disease network* ($\mathbf{A}_3$) and the *protein-protein interaction (PPI) network* ($\mathbf{A}_4$). Across those layers, we have three non-empty dependency matrices, including the *chemical-drug* dependency matrix ($\mathbf{D}_{1,2}$), the *drug-disease* interaction matrix ($\mathbf{D}_{2,3}$) and the *disease-protein* dependency matrix ($\mathbf{D}_{3,4}$) [2] .

As mentioned earlier, it is often very hard to accurately know the cross-layer dependency matrices $\{\mathbf{D}_{i,j}\ i, j = 1, ..., g\}$. In other words, such *observed* dependency

[2]More complicated dependency relationships may exist across the layers in real settings, which can be addressed with our model as well.

matrices are often incomplete and noisy. Inferring the missing cross-layer dependencies is an essential prerequisite for many multi-layered network mining tasks. On the other hand, real-world networks are evolving over time. Probing the cross-layer dependencies is often a time-consuming process in large complex networks. Thus, a newly added node could have no observed cross-layer dependencies for a fairly long period of time since its arrival. Therefore, inferring the dependencies of such kind of *zero-start* nodes is an important problem that needs to be solved efficiently. Formally, we define the cross-layer dependency inference problem (CODE) and its corresponding *zero-start* variant (CODE-ZERO) as follows.

Problem 2. (Code) Cross-Layer Dependency Inference

Given: *a multi-layered network with (1) layer-layer dependency matrix* $\mathbf{G}$*; (2) within-layer connectivity matrices* $\mathcal{A} = \{\mathbf{A}_1, ..., \mathbf{A}_g\}$*; and (3) observed cross-layer dependency matrices* $\mathcal{D} = \{\mathbf{D}_{i,j}\ i, j = 1, ..., g\}$*;*

Output: *the true cross-layer dependency matrices* $\{\tilde{\mathbf{D}}_{i,j}\ i, j = 1, ..., g\}$.

Problem 3. (Code-ZERO) Cross-Layer Dependency Inference for *zero-start* Nodes

Given: *(1) a multi-layered network* $\{\mathbf{G}, \mathcal{A}, \mathcal{D}\}$*; (2) a newly added node* p *in the* l^{th} *layer; (3) a* $1 \times n_l$ *vector* $\mathbf{s}$ *that records the connections between* p *and the existing* n_l *nodes in layer* l*;*

Output: *the true dependencies between node* p *and nodes in dependent layers of layer-*l*, i.e.,* $\tilde{\mathbf{D}}_{l,j}(p, :)$ $(j = 1, ..., g,\ \mathbf{G}(l, j) = 1)$.

4.2.2 Proposed Algorithms for CODE

In this section, we present our proposed solution for Problem 2 (CODE). We start with the proposed optimization formulation, and then present our algorithm

(FASCINATE), followed by some effectiveness and efficiency analysis.

Fascinate: Optimization Formulation

The key idea behind our formulation is to treat Problem 2 as a *collective collaborative filtering* problem. To be specific, if we view (1) nodes from a given layer (e.g., power plants) as objects from a given domain (e.g., users/items), (2) the within-layer connectivity (e.g., communication networks) as an object-object similarity measure, and (3) the cross-layer dependency (e.g., dependencies between computers in the communication layer and power plants in power grid layer) as the 'ratings' from objects of one domain to those of another domain; then inferring the missing cross-layer dependencies can be viewed as a task of inferring the missing ratings between the objects (e.g., users, items) across different domains. Having this analogy in mind, we propose to formulate Problem 2 as the following regularized optimization problem

$$\begin{aligned}\min_{\mathbf{F}_i \geq \mathbf{0}(i=1,\ldots,g)} J = & \underbrace{\sum_{i,j:\ \mathbf{G}(i,j)=1} \|\mathbf{W}_{i,j} \odot (\mathbf{D}_{i,j} - \mathbf{F}_i\mathbf{F}_j{}')\|_F^2}_{\text{C1: Matching Observed Cross-Layer Dependencies}} \\ & + \underbrace{\alpha \sum_{i=1}^{g} \operatorname{tr}(\mathbf{F}_i{}'(\mathbf{T}_i - \mathbf{A}_i)\mathbf{F}_i)}_{\text{C2: Node Homophily}} + \underbrace{\beta \sum_{i=1}^{g} \|\mathbf{F}_i\|_F^2}_{\text{C3: Regularization}}\end{aligned} \tag{4.9}$$

where $\mathbf{T}_i$ is the diagonal degree matrix of $\mathbf{A}_i$ with $\mathbf{T}_i(u,u) = \sum_{v=1}^{n_i} \mathbf{A}_i(u,v)$; $\mathbf{W}_{i,j}$ is an $n_i \times n_j$ weight matrix to assign different weights to different entries in the corresponding cross-layer dependency matrix $\mathbf{D}_{i,j}$; and $\mathbf{F}_i$ is the low-rank representation for layer i. For now, we set the weight matrices as follows: $\mathbf{W}_{i,j}(u,v)$ $= 1$ if $\mathbf{D}_{i,j}(u,v)$ is observed, and $\mathbf{W}_{i,j}(u,v) \in [0,1]$ if $\mathbf{D}_{i,j}(u,v)$ $= 0$ (i.e., unobserved). To simplify the computation, we set the weights of all unobserved entries to a global value w. We will discuss alternative choices for the weight matrices as the variants of FASCINATE later this section.

In this formulation (Eq. (4.9)), we can think of $\mathbf{F}_i$ as the low-rank representations/features of the nodes in layer i in some latent space, which is shared among different layers. The cross-layer dependencies between the nodes from two dependent layers can be viewed as the inner product of their latent features. Therefore, the intuition of the first term (i.e., C1) is that we want to match all the cross-layer dependencies, calibrated by the weight matrix $\mathbf{W}_{i,j}$. The second term (i.e., C2) is used to achieve node homophily, which says that for a pair of nodes u and v from the same layer (say layer-i), their low-rank representations should be similar (i.e., small $\|\mathbf{F}_i(u,:) - \mathbf{F}_i(v,:)\|_2$) if the within-layer connectivity between these two nodes is strong (i.e., large $\mathbf{A}_i(u,v)$). The third term (i.e., C3) is to regularize the norm of the low-rank matrices $\{\mathbf{F}_i\}_{i=1,\ldots,g}$ to prevent over-fitting.

Once we solve Eq. (4.9), for a given node u from layer-i and a node v from layer-j, the cross-layer dependency between them can be estimated as $\tilde{\mathbf{D}}_{i,j}(u,v) = \mathbf{F}_i(u,:)\mathbf{F}_j(v,:)'$.

Fascinate: Optimization Algorithm

The optimization problem defined in Eq. (4.9) is non-convex. Thus, we seek to find a local optima by the block coordinate descent method, where each $\mathbf{F}_i$ naturally forms a 'block'. To be specific, if we fix all other $\mathbf{F}_j (j = 1, \ldots, g, j \neq i)$ and ignore the constant terms, Eq. (4.9) can be simplified as

$$\underset{\mathbf{F}_i \geq \mathbf{0}}{J_i} = \sum_{j:\ \mathbf{G}(i,j)=1} \|\mathbf{W}_{i,j} \odot (\mathbf{D}_{i,j} - \mathbf{F}_i\mathbf{F}_j{}')\|_F^2 + \alpha \mathrm{tr}(\mathbf{F}_i{}'(\mathbf{T}_i - \mathbf{A}_i)\mathbf{F}_i) + \beta\|\mathbf{F}_i\|_F^2 \tag{4.10}$$

The derivative of J_i w.r.t. $\mathbf{F}_i$ is

$$\begin{aligned}\frac{\partial J_i}{\partial \mathbf{F}_i} =& 2(\sum_{j:\ \mathbf{G}(i,j)=1} [-(\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot \mathbf{D}_{i,j})\mathbf{F}_j + (\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot (\mathbf{F}_i\mathbf{F}_j{}'))\mathbf{F}_j] \\ &+ \alpha\mathbf{T}_i\mathbf{F}_i - \alpha\mathbf{A}_i\mathbf{F}_i + \beta\mathbf{F}_i)\end{aligned} \tag{4.11}$$

A fixed-point solution of Eq. (4.11) with non-negativity constraint on $\mathbf{F}_i$ leads to the following multiplicative updating rule for $\mathbf{F}_i$

$$\mathbf{F}_i(u,v) \leftarrow \mathbf{F}_i(u,v)\sqrt{\frac{\mathbf{X}(u,v)}{\mathbf{Y}(u,v)}} \tag{4.12}$$

where

$$\mathbf{X} = \sum_{j:\ \mathbf{G}(i,j)=1} (\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot \mathbf{D}_{i,j})\mathbf{F}_j + \alpha\mathbf{A}_i\mathbf{F}_i \tag{4.13}$$

$$\mathbf{Y} = \sum_{j:\ \mathbf{G}(i,j)=1} (\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot (\mathbf{F}_i\mathbf{F}_j{}'))\mathbf{F}_j + \alpha\mathbf{T}_i\mathbf{F}_i + \beta\mathbf{F}_i$$

Recall that we set $\mathbf{W}_{i,j}(u,v) = 1$ when $\mathbf{D}_{i,j}(u,v) > 0$, and $\mathbf{W}_{i,j}(u,v) = w$ when $\mathbf{D}_{i,j}(u,v) = 0$. Here, we define $\mathbf{I}^O_{i,j}$ as an indicator matrix for the observed entries in $\mathbf{D}_{i,j}$, that is, $\mathbf{I}^O_{i,j}(u,v) = 1$ if $\mathbf{D}_{i,j}(u,v) > 0$, and $\mathbf{I}^O_{i,j}(u,v) = 0$ if $\mathbf{D}_{i,j}(u,v) = 0$. Then, the estimated dependencies over the observed data can be represented as $\tilde{\mathbf{R}}_{i,j} = \mathbf{I}^O_{i,j} \odot (\mathbf{F}_i\mathbf{F}_j)$. With these notations, we can further simplify the update rule in Eq. (4.13) as follows

$$\mathbf{X} = \sum_{j:\ \mathbf{G}(i,j)=1} \mathbf{D}_{i,j}\mathbf{F}_j + \alpha\mathbf{A}_i\mathbf{F}_i \tag{4.14}$$

$$\mathbf{Y} = \sum_{j:\ \mathbf{G}(i,j)=1} ((1-w^2)\tilde{\mathbf{R}}_{i,j} + w^2\mathbf{F}_i\mathbf{F}_j{}')\mathbf{F}_j + \alpha\mathbf{T}_i\mathbf{F}_i + \beta\mathbf{F}_i \tag{4.15}$$

The proposed FASCINATE algorithm is summarized in Algorithm 5. First, it randomly initializes the low-rank matrices for each layer (step 1 - step 3). Then, it begins the iterative update procedure. In each iteration (step 4 - step 10), the algorithm alternatively updates $\{\mathbf{F}_i\}_{i=1,\ldots,g}$ one by one. We use two criteria to terminate the iteration: (1) either the Frobenius norm between two successive iterations for all $\{\mathbf{F}_i\}_{i=1,\ldots,g}$ is less than a threshold ξ, or (2) the maximum iteration number t is reached.

Algorithm 5 The FASCINATE Algorithm

Input: (1) a multi-layered network with (a) layer-layer dependency matrix $\mathbf{G}$, (b) within-layer connectivity matrices $\mathcal{A} = \{\mathbf{A}_1, ..., \mathbf{A}_g\}$, and (c) observed cross-layer node dependency matrices $\mathcal{D} = \{\mathbf{D}_{i,j}\ i, j = 1, ..., g\}$; (2) the rank size r; (3) weight w; (4) regularized parameters α and β;

Output: low-rank representations for each layer $\{\mathbf{F}_i\}_{i=1,...,g}$

for $i = 1$ to g **do**
  initialized $\mathbf{F}_i$ as $n_i \times r$ non-negative random matrix
end for
while not converge **do**
  for $i = 1$ to g **do**
    compute $\mathbf{X}$ as Eq. (4.14)
    compute $\mathbf{Y}$ as Eq. (4.15)
    update $\mathbf{F}_i$ as Eq. (4.12)
  end for
end while
return $\{\mathbf{F}_i\}_{i=1,...,g}$

Proof and Analysis

Here, we analyze the proposed FASCINATE algorithm in terms of its effectiveness as well as its efficiency.

Effectiveness Analysis. In terms of effectiveness, we show that the proposed FASCINATE algorithm indeed finds a local optimal solution to Eq. (4.9). To see this, we first give the following theorem, which says that the fixed point solution of Eq. (4.12) satisfies the *KKT* condition.

Theorem 1. *The fixed point solution of Eq.* (4.12) *satisfies the KKT condition.*

Proof. The Lagrangian function of Eq. (4.10) can be written as

$$L_i = \sum_{j:\ \mathbf{G}(i,j)=1} \|\mathbf{W}_{i,j} \odot (\mathbf{D}_{i,j} - \mathbf{F}_i{\mathbf{F}_j}')\|_F^2 \tag{4.16}$$
$$+ \alpha \mathrm{tr}({\mathbf{F}_i}'\mathbf{T}_i\mathbf{F}_i) - \alpha \mathrm{tr}({\mathbf{F}_i}'\mathbf{A}_i\mathbf{F}_i) + \beta\|\mathbf{F}_i\|_F^2 - \mathrm{tr}(\mathbf{\Lambda}'\mathbf{F}_i)$$

where $\mathbf{\Lambda}$ is the Lagrange multiplier. Setting the derivative of L_i w.r.t. $\mathbf{F}_i$ to 0, we get

$$2(\sum_{j:\ \mathbf{G}(i,j)=1} [-(\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot \mathbf{D}_{i,j})\mathbf{F}_j + (\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot (\mathbf{F}_i{\mathbf{F}_j}'))\mathbf{F}_j] \tag{4.17}$$
$$+ \alpha\mathbf{T}_i\mathbf{F}_i - \alpha\mathbf{A}_i\mathbf{F}_i + \beta\mathbf{F}_i) = \mathbf{\Lambda}$$

By the KKT complementary slackness condition, we have

$$[\underbrace{\sum_{j:\ \mathbf{G}(i,j)=1} (\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot (\mathbf{F}_i{\mathbf{F}_j}'))\mathbf{F}_j + \alpha\mathbf{T}_i\mathbf{F}_i + \beta\mathbf{F}_i}_{\mathbf{Y}} \tag{4.18}$$
$$- (\underbrace{\sum_{j:\ \mathbf{G}(i,j)=1} (\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot \mathbf{D}_{i,j})\mathbf{F}_j + \alpha\mathbf{A}_i\mathbf{F}_i)}_{\mathbf{X}}](u,v)\mathbf{F}_i(u,v) = 0$$

Therefore, we can see that the fixed point solution of Eq. (4.12) satisfies the above equation. □

The convergence of the proposed FASCINATE algorithm is given by the following lemma.

Lemma 3. *Under the updating rule in Eq.* (4.12)*, the objective function in Eq.* (4.10) *decreases monotonically.*

Proof. By expending the Frobenius norms and dropping constant terms, Eq. (4.10) can be further simplified as

$$J_i = \sum_{\mathbf{G}_{(i,j)}=1} (\underbrace{-2\mathrm{tr}((\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot \mathbf{D}_{i,j})\mathbf{F}_j\mathbf{F}_i')}_{T_1} + \underbrace{\mathrm{tr}((\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot (\mathbf{F}_i\mathbf{F}_j'))\mathbf{F}_j\mathbf{F}_i')}_{T_2}) \tag{4.19}$$

$$+ \underbrace{\alpha\mathrm{tr}(\mathbf{F}_i'\mathbf{T}_i\mathbf{F}_i)}_{T_3} \underbrace{-\alpha\mathrm{tr}(\mathbf{F}_i'\mathbf{A}\mathbf{F}_i)}_{T_4} + \underbrace{\beta\mathrm{tr}(\mathbf{F}_i\mathbf{F}_i')}_{T_5}$$

Following the auxiliary function approach in Lee and Seung (2001), the auxiliary function $H(\mathbf{F}_i, \tilde{\mathbf{F}}_i)$ of J_i must satisfy

$$H(\mathbf{F}_i, \mathbf{F}_i) = J_i, \;\; H(\mathbf{F}_i, \tilde{\mathbf{F}}_i) \geq J_i \tag{4.20}$$

Define

$$\mathbf{F}_i^{(t+1)} = \arg\min_{\mathbf{F}_i} H(\mathbf{F}_i, \mathbf{F}_i^{(t)}) \tag{4.21}$$

by this construction, we have

$$J_i^{(t)} = H(\mathbf{F}_i^{(t)}, \mathbf{F}_i^{(t)}) \geq H(\mathbf{F}_i^{(t+1)}, \mathbf{F}_i^{(t)}) \geq J_i^{(t+1)} \tag{4.22}$$

which proves that $J_i^{(t)}$ decreases monotonically.

Next, we prove that (1) we can find an auxiliary function that satisfies the above constraints and (2) the updating rule in Eq. (4.12) leads to global minimum solution to the auxiliary function.

First, we show that the following function is one of the auxiliary function of Eq. (4.19).

$$H(\mathbf{F}_i, \tilde{\mathbf{F}}_i) = \sum_{\mathbf{G}_{(i,j)}=1} (T_1' + T_2') + T_3' + T_4' + T_5' \tag{4.23}$$

where

$$T_1' = -2\sum_{u=1}^{n_i}\sum_{k=1}^{r}[(\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot \mathbf{D}_{i,j})\mathbf{F}_j](u,k)\tilde{\mathbf{F}}_i(u,k)(1 + \log\frac{\mathbf{F}_i(u,k)}{\tilde{\mathbf{F}}_i(u,k)}) \tag{4.24}$$

$$T_2' = \sum_{u=1}^{n_i}\sum_{k=1}^{r}\frac{[(\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot (\tilde{\mathbf{F}}_i\mathbf{F}_j'))\mathbf{F}_j](u,k)\mathbf{F}_i^2(u,k)}{\tilde{\mathbf{F}}_i(u,k)} \tag{4.25}$$

$$T_3' = \sum_{u=1}^{n_i}\sum_{k=1}^{r}\frac{[\alpha\mathbf{T}_i\tilde{\mathbf{F}}_i](u,k)\mathbf{F}_i^2(u,k)}{\tilde{\mathbf{F}}_i(u,k)} \tag{4.26}$$

$$T_4' = -\sum_{u=1}^{n_i}\sum_{v=1}^{n_i}\sum_{k=1}^{r}\alpha\mathbf{A}_i(u,v)\tilde{\mathbf{F}}_i(v,k)\tilde{\mathbf{F}}_i(u,k)(1 + \log\frac{\mathbf{F}_i(v,k)\mathbf{F}_i(u,k)}{\tilde{\mathbf{F}}_i(v,k)\tilde{\mathbf{F}}_i(u,k)}) \tag{4.27}$$

$$T_5' = \sum_{u=1}^{n_i}\sum_{k=1}^{r}\beta\mathbf{F}_i^2(u,k) \tag{4.28}$$

Here, we prove that $T_i' \geq T_i$ for $i = 1, \ldots, 5$ term by term.

Using the inequality $z \geq 1 + \log z$, we have

$$T_1' \geq -2\sum_{u=1}^{n_i}\sum_{k=1}^{r}[(\mathbf{W}_{i,j} \odot \mathbf{W}_{i,j} \odot \mathbf{D}_{i,j})\mathbf{F}_j](u,k)\mathbf{F}_i(u,k) = T_1 \tag{4.29}$$

$$T_4' \geq -\sum_{u=1}^{n_i}\sum_{v=1}^{n_i}\sum_{k=1}^{r}\alpha\mathbf{A}_i(u,v)\mathbf{F}_i(v,k)\mathbf{F}_i(u,k) = T_4$$

Expanding T_2', we can rewrite it as

$$T_2' = \sum_{u=1}^{n_i}\sum_{v=1}^{n_j}\sum_{k=1}^{r}\sum_{l=1}^{r}\frac{\mathbf{W}_{i,j}^2(u,v)\tilde{\mathbf{F}}_i(u,l)\mathbf{F}_j'(l,v)\mathbf{F}_j(v,k)\mathbf{F}_i^2(u,k)}{\tilde{\mathbf{F}}_i(u,k)} \tag{4.30}$$

Let $\mathbf{F}_i(u,k) = \tilde{\mathbf{F}}_i(u,k)\mathbf{Q}_i(u,k)$, then

$$\begin{aligned}
T_2' =& \sum_{u=1}^{n_i}\sum_{v=1}^{n_j}\sum_{k=1}^{r}\sum_{l=1}^{r}\mathbf{W}_{i,j}^2(u,v)\mathbf{F}_j'(l,v)\mathbf{F}_j(v,k)\tilde{\mathbf{F}}_i(u,l)\tilde{\mathbf{F}}_i(u,k)\mathbf{Q}_i^2(u,k) \qquad (4.31)\\
=& \sum_{u=1}^{n_i}\sum_{v=1}^{n_j}\sum_{k=1}^{r}\sum_{l=1}^{r}\mathbf{W}_{i,j}^2(u,v)\mathbf{F}_j'(l,v)\mathbf{F}_j(v,k)\tilde{\mathbf{F}}_i(u,l)\tilde{\mathbf{F}}_i(u,k)(\frac{\mathbf{Q}_i^2(u,k) + \mathbf{Q}_i^2(u,l)}{2})\\
\geq& \sum_{u=1}^{n_i}\sum_{v=1}^{n_j}\sum_{k=1}^{r}\sum_{l=1}^{r}\mathbf{W}_{i,j}^2(u,v)\mathbf{F}_j'(l,v)\mathbf{F}_j(v,k)\tilde{\mathbf{F}}_i(u,l)\tilde{\mathbf{F}}_i(u,k)\mathbf{Q}_i(u,k)\mathbf{Q}_i(u,l)\\
=& \sum_{u=1}^{n_i}\sum_{v=1}^{n_j}\sum_{k=1}^{r}\sum_{l=1}^{r}\mathbf{W}_{i,j}^2(u,v)\mathbf{F}_i(u,l)\mathbf{F}_j'(l,v)\mathbf{F}_j(v,k)\mathbf{F}_i(u,k)\\
=& T_2
\end{aligned}$$

For T_3', by using the following inequality in Ding *et al.* (2006)

$$\sum_{i=1}^{n}\sum_{p=1}^{k}\frac{[\mathbf{A}\mathbf{S}^*\mathbf{B}](i,p)\mathbf{S}^2(i,p)}{\mathbf{S}^*(i,p)} \geq \mathrm{tr}(\mathbf{S}'\mathbf{A}\mathbf{S}\mathbf{B}) \tag{4.32}$$

where $\mathbf{A} \in \mathbb{R}_+^{n\times n}$, $\mathbf{B} \in \mathbb{R}_+^{k\times k}$, $\mathbf{S} \in \mathbb{R}_+^{n\times k}$, $\mathbf{S}^* \in \mathbb{R}_+^{n\times k}$, and $\mathbf{A}$, $\mathbf{B}$ are symmetric, we have

$$T_3' \geq \alpha\mathrm{tr}(\mathbf{F}_i'\mathbf{T}_i\mathbf{F}_i) = T_3 \tag{4.33}$$

For T_5', we have $T_5' = T_5$. Putting the above inequalities together, we have $H(\mathbf{F}_i, \tilde{\mathbf{F}}_i) \geq J_i^s(\mathbf{F}_i)$.

Next, we find the global minimum solution to $H(\mathbf{F}_i, \tilde{\mathbf{F}}_i)$. The gradient of $H(\mathbf{F}_i, \tilde{\mathbf{F}}_i)$ is

$$\begin{aligned}\frac{1}{2}\frac{\partial H(\mathbf{F}_i, \tilde{\mathbf{F}}_i)}{\partial \mathbf{F}_i(u,k)} =& -\frac{[(\mathbf{W}_{i,j}\odot\mathbf{W}_{i,j}\odot\mathbf{D}_{i,j})\mathbf{F}_j](u,k)\tilde{\mathbf{F}}_i(u,k)}{\mathbf{F}_i(u,k)} \\ &+ \frac{[(\mathbf{W}_{i,j}\odot\mathbf{W}_{i,j}\odot(\tilde{\mathbf{F}}_i\mathbf{F}_j'))\mathbf{F}_j](u,k)\mathbf{F}_i(u,k)}{\tilde{\mathbf{F}}_i(u,k)} \\ &+ \frac{[\alpha\mathbf{T}_i\tilde{\mathbf{F}}_i](u,k)\mathbf{F}_i(u,k)}{\tilde{\mathbf{F}}_i(u,k)} - \frac{[\alpha\mathbf{A}_i\tilde{\mathbf{F}}_i](u,k)\tilde{\mathbf{F}}_i(u,k)}{\mathbf{F}_i(u,k)} + \beta\mathbf{F}_i(u,k)\end{aligned} \tag{4.34}$$

From the gradient of $H(\mathbf{F}_i, \tilde{\mathbf{F}}_i)$, we can easily get its Hessian matrix, which is a positive diagonal matrix. Therefore, the global minimum of $H(\mathbf{F}_i, \tilde{\mathbf{F}}_i)$ can be obtained by setting its gradient Eq. (4.34) to zero, which leads to

$$\mathbf{F}_i^2(u,k) = \tilde{\mathbf{F}}_i^2(u,k)\frac{[(\mathbf{W}_{i,j}\odot\mathbf{W}_{i,j}\odot(\tilde{\mathbf{F}}_i\mathbf{F}_j'))\mathbf{F}_j + \alpha\mathbf{A}_i\tilde{\mathbf{F}}_i](u,k)}{[(\mathbf{W}_{i,j}\odot\mathbf{W}_{i,j}\odot\mathbf{D}_{i,j})\mathbf{F}_j + \alpha\mathbf{T}_i\tilde{\mathbf{F}}_i + \beta\tilde{\mathbf{F}}_i](u,k)} \tag{4.35}$$

Recall that we have set $\mathbf{F}_i^{(t+1)} = \mathbf{F}_i$ and $\mathbf{F}_i^{(t)} = \tilde{\mathbf{F}}_i$. The above equation proves that the updating rule in Eq. (4.10) decreases monotonically. □

According to Theorem 1 and Lemma 3, we conclude that Algorithm 5 converges to a local minima solution for Eq. 4.10 w.r.t. each individual $\mathbf{F}_i$.

Efficiency Analysis. In terms of efficiency, we analyze both the time complexity as well as the space complexity of the proposed FASCINATE algorithm, which are summarized in Lemma 19 and Lemma 20. We can see that FASCINATE scales linearly w.r.t. the size of the entire multi-layered network.

Lemma 4. *The time complexity of Algorithm 5 is* $O([\sum_{i=1}^{g}(\sum_{j:\ \mathbf{G}_{(i,j)}=1}(m_{i,j}r + (n_i + n_j)r^2) + m_i r)]t)$.

Proof. In each iteration in Algorithm 5 for updating $\mathbf{F}_i$, the complexity of calculating $\mathbf{X}$ by Eq. (4.14) is $O(\sum_{j:\ \mathbf{G}(i,j)=1} m_{i,j}r + m_i r)$ due to the sparsity of $\mathbf{D}_{i,j}$ and $\mathbf{A}_i$. The complexity of computing $\tilde{\mathbf{R}}_{i,j}$ in $\mathbf{Y}$ is $O(m_{i,j}r)$. Computing $\mathbf{F}_i(\mathbf{F}_j'\mathbf{F}_j)$ requires $O((n_i + n_j)r^2)$ operations and computing $\alpha\mathbf{T}_i\mathbf{F}_i + \beta\mathbf{F}_i$ requires $O(n_i r)$ operations. So, it is of $O(\sum_{j:\ \mathbf{G}(i,j)=1}(m_{i,j}r + (n_i + n_j)r^2))$ complexity to get $\mathbf{Y}$ in step 7. Therefore, it takes $O(\sum_{j:\ \mathbf{G}(i,j)=1}(m_{i,j}r + (n_i + n_j)r^2) + m_i r)$ to update $\mathbf{F}_i$. Putting all together, the complexity of updating all low-rank matrices in each iteration is $O(\sum_{i=1}^{g}$ $(\sum_{j:\ \mathbf{G}(i,j)=1}(m_{i,j}r + (n_i + n_j)r^2) + m_i r))$. Thus, the overall complexity of Algorithm 5 is $O([\sum_{i=1}^{g}(\sum_{\mathbf{G}(i,j)=1}(m_{i,j}r + (n_i + n_j)r^2) + m_i r)]t)$, where t is the maximum number of iterations in the algorithm. $\square$

Lemma 5. *The space complexity of Algorithm 5 is* $O(\sum_{i=1}^{g}(n_i r + m_i) + \sum_{i,j:\ \mathbf{G}(i,j)=1} m_{i,j})$
.

Proof. It takes $O(\sum_{i=1}^{g} n_i r)$ to store all the low-rank matrices, and $O(\sum_{i=1}^{g} m_i + \sum_{i,j:\ \mathbf{G}(i,j)=1} m_{i,j})$ to store all the within-layer connectivity matrices and dependency matrices in the multi-layered network. To calculate $\mathbf{X}$ for $\mathbf{F}_i$, it costs $O(n_i r)$ to compute $\sum_{j:\ \mathbf{G}(i,j)=1} \mathbf{D}_{i,j}\mathbf{F}_j$ and $\alpha\mathbf{A}_i\mathbf{F}_i$. For $\mathbf{Y}$, the space cost of computing $\tilde{\mathbf{R}}_{i,j}$ and $\mathbf{F}_i(\mathbf{F}_j'\mathbf{F}_j)$ is $O(m_{i,j})$ and $O(n_i r)$ respectively. Therefore, the space complexity of calculating $\sum_{j:\ \mathbf{G}(i,j)=1}((1-w^2)\tilde{\mathbf{R}}_{i,j} + w^2\mathbf{F}_i{\mathbf{F}_j}')\mathbf{F}_j$ is $O(\max_{j:\ \mathbf{G}(i,j)=1} m_{i,j} + n_i r)$. On the other hand, the space required to compute $\alpha\mathbf{T}_i\mathbf{F}_i + \beta\mathbf{F}_i$ is $O(n_i r)$. Putting all together, the space cost of updating all low-rank matrices in each iteration is of $O(\max_{i,j:\ \mathbf{G}(i,j)=1} m_{i,j} +$

$\max_i n_i r$). Thus, the overall space complexity of Algorithm 5 is $O(\sum_{i=1}^{g}(n_i r + m_i) + \sum_{i,j:\ \mathbf{G}(i,j)=1} m_{i,j})$. $\square$

Variants

Here, we discuss some variants of the proposed FASCINATE algorithm.

Collective One-Class Collaborative Filtering. By setting $w \in (0,1)$, FASCINATE can be used to address one class collaborative filtering problem, where implicit dependencies extensively exist between nodes from different layers. Specifically, in two-layered networks, FASCINATE is reduced to *wiZAN-Dual*, a weighting-based, dual-regularized one class collaborative filtering algorithm proposed in Yao *et al.* (2014).

Multi-layered Network Clustering. By setting all the entries in the weight matrix $\mathbf{W}_{i,j}$ to 1 in Eq. (4.9), we have the following objective function

$$\min_{\mathbf{F}_i \geq \mathbf{0}(i=1,\ldots,g)} J = \sum_{i,j:\ \mathbf{G}(i,j)=1} \|\mathbf{D}_{i,j} - \mathbf{F}_i\mathbf{F}_j{}'\|_F^2 + \alpha \sum_{i=1}^{g} \mathrm{tr}(\mathbf{F}_i{}'(\mathbf{T}_i - \mathbf{A}_i)\mathbf{F}_i) + \beta \sum_{i=1}^{g} \|\mathbf{F}_i\|_F^2 \tag{4.36}$$

where $\mathbf{F}_i$ can be viewed as the cluster membership matrix for nodes in layer-i). By following similar procedure in FASCINATE, we can get the local optima of the above objective function with the following updating rule

$$\mathbf{F}_i(u,v) \leftarrow \mathbf{F}_i(u,v)\sqrt{\frac{\mathbf{X}_c(u,v)}{\mathbf{Y}_c(u,v)}} \tag{4.37}$$

where

$$\mathbf{X}_c = \sum_{j:\ \mathbf{G}(i,j)=1} \mathbf{D}_{i,j}\mathbf{F}_j + \alpha\mathbf{A}_i\mathbf{F}_i \tag{4.38}$$

$$\mathbf{Y}_c = \sum_{j:\ \mathbf{G}(i,j)=1} \mathbf{F}_i\mathbf{F}_j{}'\mathbf{F}_j + \alpha\mathbf{T}_i\mathbf{F}_i + \beta\mathbf{F}_i \tag{4.39}$$

Although in the above updating rule, we do not need to calculate $\tilde{\mathbf{R}}_{i,j}$ for $\mathbf{Y}_c$ comparing to $\mathbf{Y}$ in Eq. (4.15), the overall time complexity for the algorithm is still

$O([\sum_{i=1}^{g}(\sum_{j:\ \mathbf{G}_{(i,j)}=1}(m_{i,j}r + (n_i + n_j)r^2) + m_i r)]t)$. If we restrict ourselves to two-layered networks (i.e., $g = 2$), the above variant for FASCINATE becomes a dual regularized co-clustering algorithm Liu *et al.* (2015).

Unconstrained Fascinate. In FASCINATE, we place an non-negative constraint on the latent features $\{\mathbf{F}_i\}_{1=1...g}$ in Eq. (4.9) to pursue good interpretability and efficiency. By discarding the non-negative constraint, we have FASCINATE-UN, an unconstrained variant of FASCINATE, which can be solved with gradient descent method as shown in Algorithm 6. It first randomly initializes the low-rank matrices for each

---

Algorithm 6 The FASCINATE-UN Algorithm

Input: (1) a multi-layered network with (a) layer-layer dependency matrix $\mathbf{G}$, (b) within-layer connectivity matrices $\mathcal{A} = \{\mathbf{A}_1, ..., \mathbf{A}_g\}$, and (c) observed cross-layer node dependency matrices $\mathcal{D} = \{\mathbf{D}_{i,j}\ i, j = 1, ..., g\}$; (2) the rank size r; (3) weight w; (4) regularized parameters α and β; (5) parameters $a \in (0, 0.5)$, $b \in (0, 1)$

Output: low-rank representations for each layer $\{\mathbf{F}_i\}_{i=1,...,g}$

```
1:  for i = 1 to g do
2:      initialized F_i as n_i × r random matrix
3:  end for
4:  while not converge do
5:      for i = 1 to g do
6:          compute ∂J_i/∂F_i with Eq. (4.11)
7:          τ ←step size from backtracking line search
8:          F_i ← F_i − τ ∂J_i/∂F_i
9:      end for
10: end while
11: return {F_i}_{i=1,...,g}
```

---

layer (step 1 - step 3) and then begins the iterative update procedure. In each iteration (step 4 - step 10), the algorithm alternatively updates $\{\mathbf{F}_i\}_{i=1,\ldots,g}$ with gradient descent method one by one. Similar to FASCINATE, the two criteria we use to terminate the iteration are: (1) either the difference of the objective function (J in Eq. (4.9)) between two successive iterations is less than a threshold ξ, or (2) the maximum iteration number t is reached. The complexity of computing $\frac{\partial J_i}{\partial \mathbf{F}_i}$ is the same with the complexity of computing $\mathbf{X}$ and $\mathbf{Y}$ in Algorithm 5. However, in the backtracking line search procedure in step 7, calculating the value of the objective function J_i is required to find step size τ with complexity $O(\sum_{j:\ \mathbf{G}(i,j)=1} n_i n_j r + n_i^2 r)$. This quadratic complexity would increase the overall complexity of Algorithm 6 significantly in large systems.

Collective Matrix Factorization. Instead of exploiting node homophily effect from each layer, we can view the with-in layer networks as additional constraints for matrix factorization problem as modeled in the following objective function

$$\min_{\mathbf{F}_i \geq \mathbf{0}(i=1,\ldots,g)} \sum_{i,j:\ \mathbf{G}(i,j)=1} \|\mathbf{W}_{i,j} \odot (\mathbf{D}_{i,j} - \mathbf{F}_i \mathbf{F}_j{}')\|_F^2 + \alpha \sum_{i=1}^{g} \|\mathbf{A}_i - \mathbf{F}_i \mathbf{F}_i{}'\|_F^2 + \beta \sum_{i=1}^{g} \|\mathbf{F}_i\|_F^2 \tag{4.40}$$

where $\mathbf{F}_i$ is the latent features for nodes in layer-i).

Again, the above problem can be solved with similar procedure in FASCINATE. The updating rules are as follows.

$$\mathbf{F}_i(u,v) \leftarrow \mathbf{F}_i(u,v) \sqrt{\frac{\mathbf{X}_{col}(u,v)}{\mathbf{Y}_{col}(u,v)}} \tag{4.41}$$

where $\mathbf{X}_{col}$ and $\mathbf{Y}_{col}$ are defined as

$$\mathbf{X}_{col} = \sum_{j:\ \mathbf{G}(i,j)=1} \mathbf{D}_{i,j} \mathbf{F}_j + 2\alpha \mathbf{A}_i \mathbf{F}_i \tag{4.42}$$

$$\mathbf{Y}_{col} = \sum_{j:\ \mathbf{G}(i,j)=1} ((1-w^2)\tilde{\mathbf{R}}_{i,j} + w^2 \mathbf{F}_i \mathbf{F}_j{}')\mathbf{F}_j + 2\alpha \mathbf{F}_i \mathbf{F}_i{}' \mathbf{F}_i + \beta \mathbf{F}_i \tag{4.43}$$

The complexity of the above method is of the same order with FASCINATE. In particular, when the within-layer connectivity matrices $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_g\}$ are absent, the proposed FASCINATE can be viewed as a collective matrix factorization method in Singh and Gordon (2008).

While the proposed FASCINATE includes these existing methods as its special cases, its major advantage lies in its ability to collectively leverage all the available information (e.g., the within-layer connectivity, the observed cross-layer dependency) for dependency inference. As we will demonstrate in the experimental section, such a methodical strategy leads to a substantial and consistent inference performance boosting. Nevertheless, a largely unanswered question for these methods (including FASCINATE) is how to handle *zero-start* nodes. That is, when a new node arrives with no observed cross-layer dependencies, how can we effectively and efficiently infer its dependencies without rerunning the algorithm from scratch. In the next section, we present a *sub-linear* algorithm to solve this problem (i.e., Problem 2).

4.2.3 Proposed Algorithm for CODE-ZERO

A multi-layered network often exhibits high dynamics, e.g., the arrival of new nodes. For example, for a newly identified chemical in the biological system, we might know how it interacts with some existing chemicals (i.e., the within-layer connectivity). However, its cross-layer dependencies w.r.t. proteins and/or diseases might be completely unknown . This section addresses such *zero-start* problems (i.e., Problem 2). Without loss of generality, we assume that the newly added node resides in layer-*1*, indexed as its $(n_1 + 1)^{\text{th}}$ node. The within-layer connectivity between the newly added node and the existing n_1 nodes is represented by a $1 \times n_1$ row vector $\mathbf{s}$, where $\mathbf{s}(u)$ $(u = 1, ..., n_1)$ denotes the (within-layer) connectivity between the newly added node and the u^{th} existing node in layer-*1*.

We could just rerun our FASCINATE algorithm on the entire multi-layered network with the newly added node to get its low-rank representation (i.e., a $1\times r$ row vector $\mathbf{f}$), based on which its cross-layer dependencies can be estimated. However, the running time of this strategy is linear w.r.t. the size of the *entire* multi-layered network. For example, on a three-layered infrastructure network whose size is in the order of 14 million, it would take FASCINATE $2,500+$ seconds to update the low-rank matrices $\{\mathbf{F}_i\}$ for a *zero-start* node with rank $r = 200$, which might be too costly in online settings. In contrast, our upcoming algorithm is *sub-linear*, and it only takes less than 0.001 seconds on the same network without jeopardizing the accuracy.

There are two key ideas behind our online algorithm. The first is to view the newly added node as a perturbation to the original network. In detail, the updated within-layer connectivity matrix $\hat{\mathbf{A}}_1$ for layer-*1* can be expressed as

$$\hat{\mathbf{A}}_1 = \begin{bmatrix} \mathbf{A}_1 & \mathbf{s}' \\ \mathbf{s} & 0 \end{bmatrix} \tag{4.44}$$

where $\mathbf{A}_1$ is the within-layer connectivity matrix for layer-*1* before the arrival of the new node.

Correspondingly, the updated low-rank representation matrix for layer-*1* can be expressed as $\hat{\mathbf{F}}_1 = [\hat{\mathbf{F}}'_{1(n_1\times r)}\ \ \mathbf{f}']'$, where $\hat{\mathbf{F}}_{1(n_1\times r)}$ is the updated low-rank representation for the existing n_1 nodes in layer-*1*. Then the new objective function $\hat{J}$ in Eq. (4.9) can be reformatted as

$$\hat{J} = \sum_{\substack{i,j:\ \mathbf{G}(i,j)=1 \\ i,j\neq 1}} \|\mathbf{W}_{i,j} \odot (\mathbf{D}_{i,j} - \hat{\mathbf{F}}_i\hat{\mathbf{F}}_j')\|_F^2 + \sum_{j:\ \mathbf{G}(1,j)=1} \|\hat{\mathbf{W}}_{1,j} \odot (\hat{\mathbf{D}}_{1,j} - \hat{\mathbf{F}}_1\hat{\mathbf{F}}_j')\|_F^2 \quad (4.45)$$

$$+ \sum_{i=2}^{g} \frac{\alpha}{2} \sum_{u=1}^{n_i}\sum_{v=1}^{n_i} \mathbf{A}_i(u,v)\|\hat{\mathbf{F}}_i(u,:) - \hat{\mathbf{F}}_i(v,:)\|_2^2$$

$$+ \frac{\alpha}{2} \sum_{u=1}^{n_1}\sum_{v=1}^{n_1} \mathbf{A}_1(u,v)\|\hat{\mathbf{F}}_1(u,:) - \hat{\mathbf{F}}_1(v,:)\|_2^2$$

$$+ \beta \sum_{i=2}^{g} \|\hat{\mathbf{F}}_i\|_F^2 + \beta\|\hat{\mathbf{F}}'_{1(n_1\times r)}\|_F^2 + \alpha \sum_{v=1}^{n_1} \mathbf{s}(v)\|\mathbf{f} - \hat{\mathbf{F}}_1(v,:)\|_2^2 + \beta\|\mathbf{f}\|_2^2$$

Since the newly added node has no dependencies, we can set

$$\hat{\mathbf{W}}_{1,j} = \begin{bmatrix} \mathbf{W}_{1,j} \\ \mathbf{0}_{(1\times n_j)} \end{bmatrix}, \quad \hat{\mathbf{D}}_{1,j} = \begin{bmatrix} \mathbf{D}_{1,j} \\ \mathbf{0}_{(1\times n_j)} \end{bmatrix}$$

Therefore, the second term in $\hat{J}$ can be simplified as

$$\sum_{j:\ \mathbf{G}(1,j)=1} \|\mathbf{W}_{1,j} \odot (\mathbf{D}_{1,j} - \hat{\mathbf{F}}_{1(n_1\times r)}\hat{\mathbf{F}}_j')\|_F^2 \quad (4.46)$$

Combining Eq. (4.45), Eq. (4.46) and J in Eq. (4.9) together, $\hat{J}$ can be expressed as

$$\hat{J} = J + J^1 \quad (4.47)$$

where $J^1 = \alpha \sum_{v=1}^{n_1} \mathbf{s}(v)\|\mathbf{f} - \hat{\mathbf{F}}_1(v,:)\|_2^2 + \beta\|\mathbf{f}\|_2^2$, and J is the objective function without the newly arrived node.

The second key idea of our online algorithm is that in Eq. (4.47), J is often orders of magnitude larger than J^1. For example, in the **BIO** dataset used in Section 4.2.3, J is in the order of 10^3, while J^1 is in the order of 10^{-1}. This naturally leads to the following approximation strategy, that is, we (1) fix J with $\{\mathbf{F}_i^*\}_{i=1,\ldots,g}$ (i.e., the previous local optimal solution to Eq. (4.9) without the newly arrived node), and (2) optimize J^1 to find out the low-rank representation $\mathbf{f}$ for the newly arrived node. That is, we seek to solve the following optimization problem

$$\mathbf{f} = \arg\min_{\mathbf{f}\geq\mathbf{0}} J^1 \quad \text{subject to: } \hat{\mathbf{F}}_{1(n_1\times r)} = \mathbf{F}_1^* \tag{4.48}$$

with which, we can get an approximate solution $\{\hat{\mathbf{F}}_i\}_{i=1,\ldots,g}$ to $\hat{J}$.

To solve $\mathbf{f}$, we take the derivative of J^1 w.r.t. $\mathbf{f}$ and get

$$\begin{aligned}\frac{1}{2}\frac{\partial J^1}{\partial \mathbf{f}} &= \beta\mathbf{f} + \alpha\sum_{v=1}^{n_1}\mathbf{s}(v)(\mathbf{f} - \mathbf{F}_1^*(v,:)) \\ &= (\beta + \alpha\sum_{v=1}^{n_1}\mathbf{s}(v))\mathbf{f} - \alpha\mathbf{s}\mathbf{F}_1^*\end{aligned} \tag{4.49}$$

Since α and β are positive, the Hessian matrix of J^1 is a positive diagonal matrix. Therefore, the global minimum of J^1 can be obtained by setting its derivative to zero. Then the optimal solution to J^1 can be expressed as

$$\mathbf{f} = \frac{\alpha\mathbf{s}\mathbf{F}_1^*}{\beta + \alpha\sum_{v=1}^{n_1}\mathbf{s}(v)} \tag{4.50}$$

For the newly added node, $\mathbf{f}$ can be viewed as the weighted average of its neighbors' low-rank representations. Notice that in Eq. (4.50), the non-negativity constraint on $\mathbf{f}$ naturally holds. Therefore, we refer to this solution (i.e., Eq. (4.50)) as FASCINATE-ZERO. In this way, we can successfully decouple the cross-layer dependency inference problem for the *zero-start* node from the entire multi-layered network and localize it only among its neighbors in layer-*1*. The localization significantly reduces the time complexity, as summarized in Lemma 6, which is linear w.r.t. the number of neighbors of the new node (and therefore is *sub-linear* w.r.t. the size of the entire network).

Lemma 6. *Let* $nnz(\mathbf{s})$ *denotes the total number of within-layer links between the newly added node and the original nodes in layer-*1 *(i.e.,* $nnz(\mathbf{s})$ *is the degree for the newly added node). Then the time complexity of* FASCINATE-ZERO *is* $O(nnz(\mathbf{s})r)$.

Proof. Since the links between the newly added node and the original nodes in layer-*1* are often very sparse, the number of non-zero elements in $\mathbf{s}$ (nnz($\mathbf{s}$)) is much smaller than n_1. Therefore, the complexity of computing $\mathbf{s}\mathbf{F}_1^*$ can be reduced to $O(\text{nnz}(\mathbf{s})r)$. The multiplication between α and $\mathbf{s}\mathbf{F}_1^*$ takes $O(r)$. Computing $\sum_{v=1}^{n_1} \mathbf{s}(v)$ takes $O(\text{nnz}(\mathbf{s}))$. Thus, the overall complexity of computing $\mathbf{f}$ is $O(\text{nnz}(\mathbf{s})r)$. □

Remarks. Following the similar procedure in FASCINATE-ZERO, it is easy to extend the zero-start problem to the scenario where a new within-layer edge is added to two existing nodes. Suppose in layer-*1*, a new edge $\langle u, v\rangle$ is added between node u and node v. To find out the updated low-rank matrices $\{\hat{\mathbf{F}}_i\}$ efficiently after the perturbation, we can partition the nodes in the multi-layered network into two parts: (1) nodes that can be affected by either node u or node v (denoted as $\mathcal{N}^{\{u,v\}}$) and (2) nodes that are irrelevant to both node u and node v (denoted as $\mathcal{N}^{\backslash\{u,v\}}$). Specifically, we define that node w can be affected by node u if and only if there exists a path from u to w, and the links in the path can be either within-layer edges or cross-layer dependencies; otherwise, node w is viewed as irrelevant to u. By this definition, we have $\mathcal{N}^{\{u,v\}} \cap \mathcal{N}^{\backslash\{u,v\}} = \Phi$ and the new objective function $\hat{J}$ can be decomposed into two parts as

$$\hat{J} = \hat{J}^{\{u,v\}} + \hat{J}^{\backslash\{u,v\}} \tag{4.51}$$

where $\hat{J}^{\{u,v\}}$ only contains the optimization terms for the latent features of the affected nodes ($\{\hat{\mathbf{F}}_i\}^{\{u,v\}}$), while $\hat{J}^{\backslash\{u,v\}}$ contains the terms for latent features of irrelevant nodes ($\{\hat{\mathbf{F}}_i\}^{\backslash\{u,v\}}$). As the newly added edge $\langle u, v\rangle$ in layer-*1* would not cause any changes in $\hat{J}^{\backslash\{u,v\}}$, $\{\hat{\mathbf{F}}_i\}^{\backslash\{u,v\}}$ would remain the same with the previous local optima solution $\{\mathbf{F}_i^*\}^{\backslash\{u,v\}}$. Therefore, the only terms we need to optimize is $\hat{J}^{\{u,v\}}$ w.r.t. the affected latent features $\{\hat{\mathbf{F}}_i\}^{\{u,v\}}$.

4.2.4 Experimental Evaluation

In this section, we evaluate the proposed FASCINATE algorithms. All experiments are designed to answer the following questions:

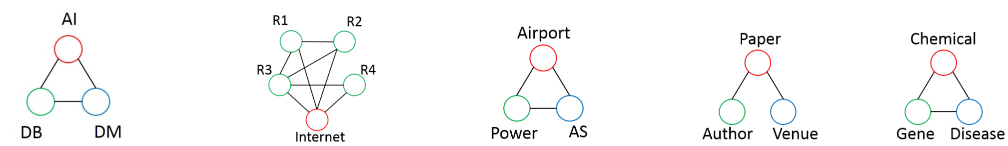
- *Effectiveness.* How effective are the proposed FASCINATE algorithms in inferring the missing cross-layer dependencies?
- *Efficiency.* How fast and scalable are the proposed algorithms?

Experimental Setup

Datasets Description. We perform our evaluations on five different datasets, including (1) a three-layer cross-domain paper citation network in the academic research domain (CITATION); (2) a five-layer Italy network in the critical infrastructure domain (INFRA-5); (3) a three-layer network in the critical infrastructure domain (INFRA-3); (4) a three-layer CTD (Comparative Toxicogenomics Database) network in the biological domain (BIO); and (5) a three-layer Aminer academic network in the social collaboration domain (SOCIAL). The statistics of these datasets are shown in Table 4.11, and the abstract layer-layer dependency graphs of these four datasets are summarized in Figure 4.18 In all these four datasets, the cross-layer dependencies are binary and undirected (i.e., $\mathbf{D}_{i,j}(u, v) = \mathbf{D}_{j,i}(v, u)$).

Table 4.3: Statistics of Datasets.

Dataset	# of Layers	# of Nodes	# of Links	# of CrossLinks
CITATION	3	33,249	27,017	4,589
INFRA-5	5	349	379	565
INFRA-3	3	15,126	29,861	28,023,500
SOCIAL	3	125,344	214,181	188,844
BIO	3	35,631	253,827	75,456



(a) CITATION (b) INFRA-5 (c) INFRA-3 (d) SOCIAL (e) BIO

Figure 4.18: The Abstract Dependency Structure of Each Dataset.

CITATION. The construction of this publication network is based on the work in Li *et al.* (2015a). It contains three layers, which correspond to the paper citation networks in AI (Artificial Intelligence), DB (Database) and DM (Data Mining) domains. The cross-domain citations naturally form the cross-layer dependencies in the system. For example, the cross-layer dependency between AI layer and DM layer indicates the citations between AI papers and DM papers. The papers in the system are from the top conferences in the corresponding areas as shown in Table 4.4. The number of nodes in each layer varies from $5,158$ to $18,243$, and the number of within-layer links ranges from $20,611$ to $40,885$. The number of cross-layer dependencies ranges from 536 to $2,250$. The structure of the entire system is shown in Figure 4.18 (a).

Table 4.4: List of Conferences in Each Domain.

Domain	**AI**	**DM**	**DB**
Conferences	IJCAI	KDD	SIGMOD
	AAAI	ICDM	VLDB
	ICML	SDM	ICDM
	NIPS	PKDD	PODS

INFRA-5. The construction of this critical infrastructure network is based on the data implicated from an electrical blackout in Italy in Sept 2003 Rosato *et al.*

(2008). It contains five layers, including four layers of regional power grids and one Internet network Rosato *et al.* (2008). The regional power grids are partitioned by macroregions [3] . To make the regional networks more balanced, we merge the Southern Italy power grid and the Island power grid together. The power transfer lines between the four regions are viewed as cross-layer dependencies. For the Italy Internet network, it is assumed that each Internet center is supported by the power stations within a radius of 70km. Its abstract dependency graph is shown in Figure 4.18(b). The smallest layer in the network has 39 nodes and 50 links; while the largest network contains 151 nodes and 158 links. The number of dependencies is up to 307.

INFRA-3. This dataset contains the following three critical infrastructure networks: an airport network [4] , an autonomous system network [5] and a power grid Watts and Strogatz (1998). We construct a three-layered network in the same way as Chen *et al.* (2015). The three infrastructure networks are functionally dependent on each other. Therefore, they form a triangle-shaped multi-layered network as shown in Figure 4.18(c). The construction of the cross-layer dependencies is based on geographic proximity.

SOCIAL. This dataset contains three layers, including a collaboration network among authors, a citation network between papers and a venue network Tang *et al.* (2008). The number of nodes in each layer ranges from 899 to $62,602$, and the number of within-layer links ranges from $2,407$ to $201,037$. The abstract layer-layer dependency graph of SOCIAL is shown in Figure 4.18(d). The collaboration layer is connected to the paper layer with the authorship dependency, while the venue layer is connected to the paper layer with publishing dependency. For the *Paper-*

[3] https://en.wikipedia.org/wiki/First-level_NUTS_of_the_European_Union

[4] http://www.levmuchnik.net/Content/Networks/NetworkData.html

[5] http://snap.stanford.edu/data/

Author dependency, we have 126,242 links across the two layers; for the *Paper-Venue* dependency, we have 62,602 links.

BIO. The construction of the CTD network is based on the works in Davis *et al.* (2015); Razick *et al.* (2008); Van Driel *et al.* (2006). It contains three layers, which are chemical, disease and gene similarity networks. The number of nodes in these networks ranges from $4,256$ to $25,349$, and the number of within-layer links ranges from $30,551$ to $154,167$. The interactions between chemicals, genes, and diseases form the cross-layer dependency network as shown in Figure 4.18(e). For *Chemical-Gene* dependency, we have 53,735 links across the two layers; for *Chemical-Disease* dependency, we have 19,771 links; and for *Gene-Disease* dependency, we have 1,950 links.

For all datasets, we randomly select 50% cross-layer dependencies as the training set and use the remaining 50% as the test set.

Comparing Methods. We compare FASCINATE with the following methods, including (1) FASCINATE-CLUST - a variant of the proposed method for the purpose of dependency clustering, (2) FASCINATE-UN - a variant of FASCINATE without non-negative constraint, (3) *MulCol* - a collective matrix factorization method Singh and Gordon (2008), (4) *PairSid* - a pairwise one-class collaborative filtering method proposed in Yao *et al.* (2014), (5) *PairCol* - a pairwise collective matrix factorization method degenerated from *MulCol* (6) *PairNMF* - a pairwise non-negative matrix factorization (*NMF*) based method Lin (2007), (7) *PairRec* - a pairwise matrix factorization based algorithm introduced in Koren *et al.* (2009), (8) *FlatNMF* - an NMF based method that treats the input multi-layered network as a flat-structured single network (i.e., by putting the within-layer connectivity matrices in the diagonal blocks, and the cross-layer dependency matrices in the off-diagonal blocks), and (9) *FlatRec* - a matrix factorization based method using the same techniques as *PairRec* but treating the input multi-layered network as a single network as in *FlatNMF*.

For the experimental results reported in this work, we set rank $r = 100$, maximum iteration $t = 100$, termination threshold $\xi = 10^{-8}$, weight $w^2 = 0.1$, regularization parameters $\alpha = 0.1$, $\beta = 0.1$ and backtracking line search parameters $a = 0.1$, $b = 0.8$ unless otherwise stated.

Evaluation Metrics. We use the following metrics for the effectiveness evaluations.

- **MAP**. It measures the mean average precision over all entities in the cross-layer dependency matrices Li *et al.* (2010). A larger *MAP* indicates better inference performance.

- **R-MPR**. It is a variant of Mean Percentage Ranking for one-class collaborative filtering Hu *et al.* (2008). *MPR* is originally used to measure the user's satisfaction of items in a ranked list. In our case, we can view the nodes from one layer as users, and the nodes of the dependent layer(s) as items. The ranked list therefore can be viewed as ordered dependencies by their importance. Smaller *MPR* indicates better inference performance. Specifically, for a randomly produced list, its *MPR* is expected to be 50%. Here, we define R-MPR = 0.5 − MPR, so that larger *R-MRP* indicates better inference performance.

- **HLU**. Half-Life Utility is also a metric from one-class collaborative filtering. By assuming that the user will view each consecutive items in the list with exponential decay of possibility, it estimates how likely a user will choose an item from a ranked list Pan *et al.* (2008). In our case, it measures how likely a node will establish dependencies with the nodes in the ranked list. A larger *HLU* indicates better inference performance.

- **AUC**. Area Under ROC Curve is a metric that measures the classification accuracy. A larger *AUC* indicates better inference performance.

- **Prec@K**. Precision at K is defined by the proportion of true dependencies among the top K inferred dependencies. A larger *Prec@K* indicates better inference performance.

Machine and Repeatability All the experiments are performed on a machine with 2 processors Intel Xeon 3.5GHz with 256GB of RAM. The algorithms are programmed with MATLAB using single thread.

Effectiveness

In this section, we aim to answer the following three questions, (1) how effective is FASCINATE for Problem 1 (i.e., CODE)? (2) how effective is FASCINATE-ZERO for Problem 2 (i.e., CODE-ZERO)? and (3) how sensitive are the proposed algorithms w.r.t. the model parameters?

Effectiveness of Fascinate. We compare the proposed algorithms and the existing methods on all the five datasets. As FASCINATE-UN is not scalable to large networks, we only evaluate its performance on two small datasets—CITATION and INFRA-5. The results are shown in Table 4.5 through Table 4.6. There are several interesting observations. First is that our proposed FASCINATE algorithm and its variants (FASCINATE-CLUST and FASCINATE-UN) consistently outperform all other methods in terms of all the five evaluation metrics. Second, by exploiting the structure of multi-layered network, FASCINATE, FASCINATE-CLUST, FASCINATE-UN, and *MulCol* can achieve significantly better performance than the pairwise methods in most datasets. Third, among the pairwise baselines, *PairSid* and *PairCol* are better than *PairNMF* and *PairRec*. The main reason is that the first two algorithms utilize both within-layer connectivity matrices and cross-layer dependency matrix for matrix factorization, while the latter two only use the observed dependency matrix. Finally, the relatively poor performance of *FlatNMF* and *FlatRec* implies that simply

Table 4.5: Cross-Layer Dependency Inference on CITATION.

Methods	MAP	R-MPR	HLU	AUC	Prec@10
FASCINATE	0.1389	**0.3907**	19.1264	**0.8523**	0.0428
FASCINATE-CLUST	0.1347	0.3882	19.8367	0.8487	0.0407
FASCINATE-UN	**0.1873**	0.2685	**25.1961**	0.7423	**0.0532**
MulCol	0.1347	0.3882	19.8367	0.8487	0.0459
PairSid	0.1623	0.3868	21.8641	0.8438	0.0480
PairCol	0.1311	0.3838	19.1697	0.8388	0.0446
PairNMF	0.0338	0.1842	4.4397	0.6009	0.0103
PairRec	0.0351	0.2582	5.3407	0.6527	0.0129
FlatNMF	0.0811	0.3539	12.1835	0.8084	0.0284
FlatRec	0.0032	0.3398	0.0608	0.8113	0.0001

flattening the multi-layered network into a single network is insufficient to capture the intrinsic correlations across different layers.

We also test the sensitivity of the proposed algorithms w.r.t. the sparsity of the observed cross-layer dependency matrices (i.e., the ratio of the missing values) on INFRA-3. The results in Figure 4.19 demonstrate that both FASCINATE and FASCINATE-CLUST perform well even when 90%+ entries in the dependency matrices are missing.

Effectiveness of Fascinate-ZERO. To evaluate the effectiveness of FASCINATE-ZERO, we randomly select one node from the *Chemical* layer in the BIO dataset as the newly arrived node and compare the inference performance between FASCINATE-ZERO and FASCINATE. The average results over multiple runs are presented in Figure 4.20. We can see that FASCINATE-ZERO bears a very similar inference power as FASCINATE, but it is orders of magnitude faster. We observe similar performance when the *zero-start* nodes are selected from the other two layers (i.e., *Gene* and

Table 4.6: Cross-Layer Dependency Inference on INFRA-5

Methods	MAP	R-MPR	HLU	AUC	Prec@10
Fascinate	**0.5040**	**0.3777**	**67.2231**	**0.8916**	**0.2500**
Fascinate-Clust	0.4297	0.3220	56.8215	0.8159	0.2340
Fascinate-UN	0.4354	0.3631	60.2393	0.8575	0.2412
MulCol	0.4523	0.3239	59.8115	0.8329	0.2413
PairSid	0.3948	0.2392	49.5484	0.7413	0.2225
PairCol	0.3682	0.2489	48.5966	0.7406	0.2309
PairNMF	0.1315	0.0464	15.7148	0.5385	0.0711
PairRec	0.0970	0.0099	9.4853	0.5184	0.0399
FlatNMF	0.3212	0.2697	44.4654	0.7622	0.1999
FlatRec	0.1020	0.0778	11.5598	0.5740	0.0488

Table 4.7: Cross-Layer Dependency Inference on INFRA-3.

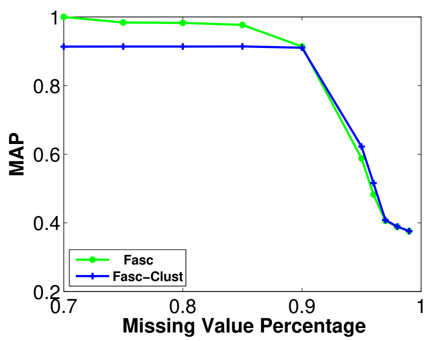
Methods	MAP	R-MPR	HLU	AUC	Prec@10
Fascinate	0.4780	0.0788	**55.7289**	0.6970	**0.5560**
Fascinate-Clust	**0.5030**	**0.0850**	49.1223	**0.7122**	0.4917
Fascinate-UN	-	-	-	-	-
MulCol	0.4606	0.0641	49.3585	0.6706	0.4930
PairSid	0.4253	0.0526	47.7284	0.5980	0.4773
PairCol	0.4279	0.0528	48.1314	0.5880	0.4816
PairNMF	0.4275	0.0511	48.8478	0.5579	0.4882
PairRec	0.3823	0.0191	38.9226	0.5756	0.3895
FlatNMF	0.4326	0.0594	45.0090	0.6333	0.4498
FlatRec	0.3804	0.0175	38.0550	0.5740	0.3805

Table 4.8: Cross-Layer Dependency Inference on SOCIAL

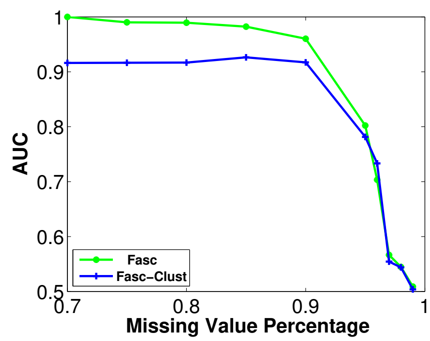
Methods	**MAP**	**R-MPR**	**HLU**	**AUC**	**Prec@10**
Fascinate	0.0660	**0.2651**	**8.4556**	**0.7529**	**0.0118**
Fascinate-Clust	**0.0667**	0.2462	8.2160	0.7351	0.0108
Fascinate-UN	-	-	-	-	-
MulCol	0.0465	0.2450	6.0024	0.7336	0.0087
PairSid	0.0308	0.1729	3.8950	0.6520	0.0062
PairCol	0.0303	0.1586	3.7857	0.6406	0.0056
PairNMF	0.0053	0.0290	0.5541	0.4998	0.0007
PairRec	0.0056	0.0435	0.5775	0.5179	0.0007
FlatNMF	0.0050	0.0125	0.4807	0.5007	0.0007
FlatRec	0.0063	0.1009	0.6276	0.5829	0.0009

Table 4.9: Cross-Layer Dependency Inference on Bio.

Methods	**MAP**	**R-MPR**	**HLU**	**AUC**	**Prec@10**
Fascinate	**0.3979**	**0.4066**	**45.1001**	**0.9369**	**0.1039**
Fascinate-Clust	0.3189	0.3898	37.4089	0.9176	0.0857
Fascinate-UN	-	-	-	-	-
MulCol	0.3676	0.3954	42.8687	0.9286	0.0986
PairSid	0.3623	0.3403	40.4048	0.8682	0.0941
PairCol	0.3493	0.3153	38.4364	0.8462	0.0889
PairNMF	0.1154	0.1963	15.8486	0.6865	0.0393
PairRec	0.0290	0.2330	3.6179	0.7105	0.0118
FlatNMF	0.2245	0.2900	26.1010	0.8475	0.0615
FlatRec	0.0613	0.3112	8.4858	0.8759	0.0254



(a) MAP



(b) AUC

Figure 4.19: Performance of FASCINATE and FASCINATE-CLUST on INFRA-3 Dataset under Different Missing Value Percentages.

Disease).

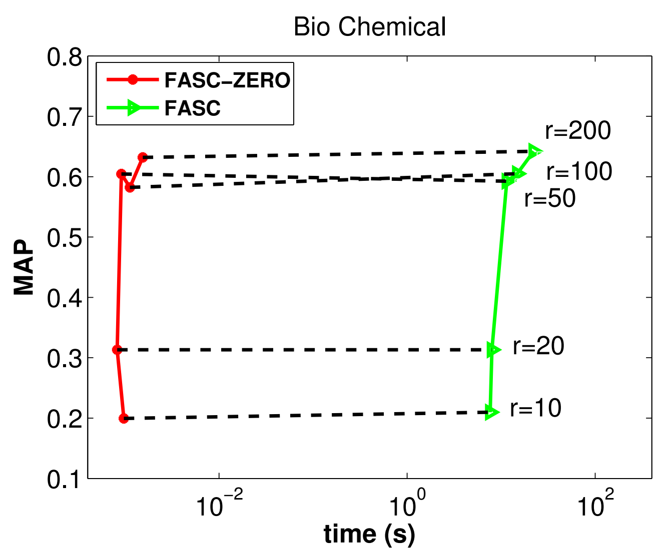
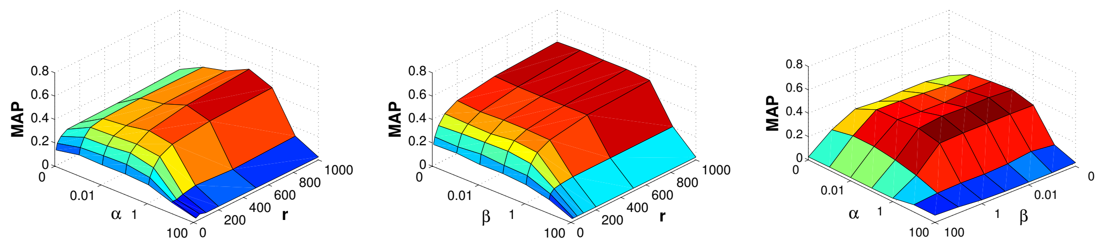


Figure 4.20: The Effectiveness of FASCINATE-ZERO in BIO Network W.R.T. Different Rank r.

Parameter Studies. There are three parameters α, β, and r in the proposed FASCINATE algorithm. α is used to control the impact of node homophily, β is used to avoid over-fitting, and r is the number of columns of the low-rank matrices $\{\mathbf{F}_i\}$. We fix one of these parameters, and study the impact of the remaining two on the inference results. From Figure 4.21, we can see that *MAP* is stable over a wide range of both α and β. As for the third parameter r, the inference performance quickly increases w.r.t. r until it hits 200, after which the MAP is almost flat. This suggests that relatively small size of the low-rank matrices might be sufficient to achieve a satisfactory inference performance.



(a) Impact of α and r. (fixing $\beta = 0.1$)

(b) Impact of β and r. (fixing $\alpha = 0.1$)

(c) Impact of α and β . (fixing $r = 100$)

Figure 4.21: The Parameter Studies of the Bio Dataset.

For Fascinate-UN, we study the impact of the backtracking line search parameters on its performance. By fixing α, β and rank r to 0.1, 0.1 and 100 respectively, we examine a wide range of a and b within their domains as shown in Figure 4.22. We can see that the inference performance is sensitive to the combination of a and b because subtle parameter changes may affect the convergence speed in Algorithm 6 greatly, which would have impact on the inference performance within limited iterations consequently.

Efficiency

The scalability results of Fascinate and Fascinate-ZERO are presented in Figure 4.23. As we can see in Figure 4.23(a), Fascinate scales linearly w.r.t. the overall network size (i.e., $\sum_i (n_i + m_i) + \sum_{i,j} m_{i,j}$), which is consistent with our previous analysis in Lemma 19. As for Fascinate-ZERO, it scales *sub-linearly* w.r.t. the entire network size. This is because, by Lemma 6, the running time of Fascinate-ZERO is

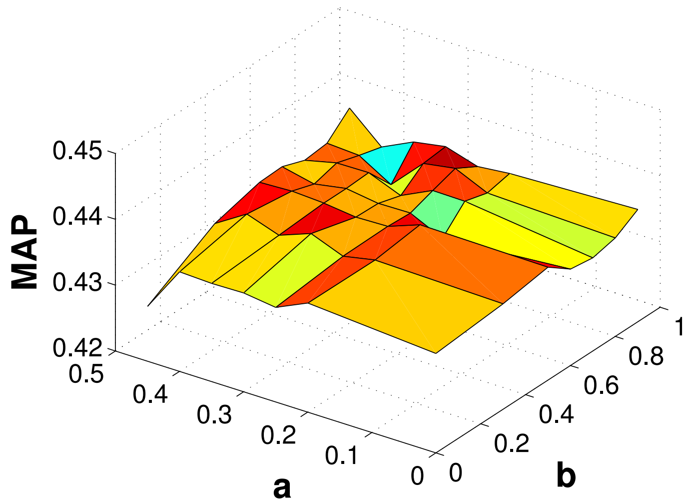
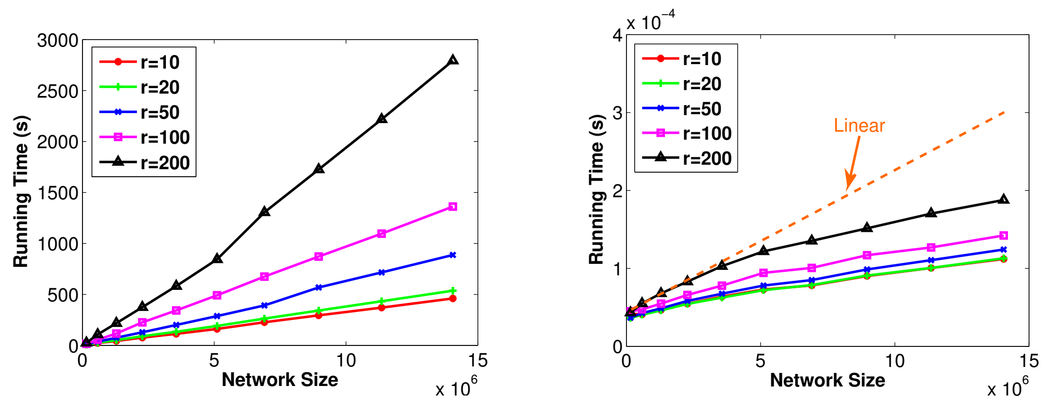


Figure 4.22: The Backtracking Line Search Parameter Study of the INFRA-5 Dataset.

only dependent on the neighborhood size of the newly added node, rather than that of the entire network. Finally, we can see that FASCINATE-ZERO is much more efficient than FASCINATE. To be specific, on the entire INFRA-3 dataset, FASCINATE-ZERO is $10,000,000+$ faster than FASCINATE (i.e., 1.878×10^{-4} seconds vs. 2.794×10^{3} seconds).



(a) FASCINATE (b)FASCINATE-ZERO

Figure 4.23: Wall-clock Time vs. the Size of the Network.

In addition, we compare the running time of FASCINATE and FASCINATE-UN on

CITATION and INFRA-5 networks. The results are as shown in Figure 4.24. As we can see, FASCINATE-UN is orders of magnitude slower than FASCINATE to achieve similar inference results, which is consistent with our complexity analysis.

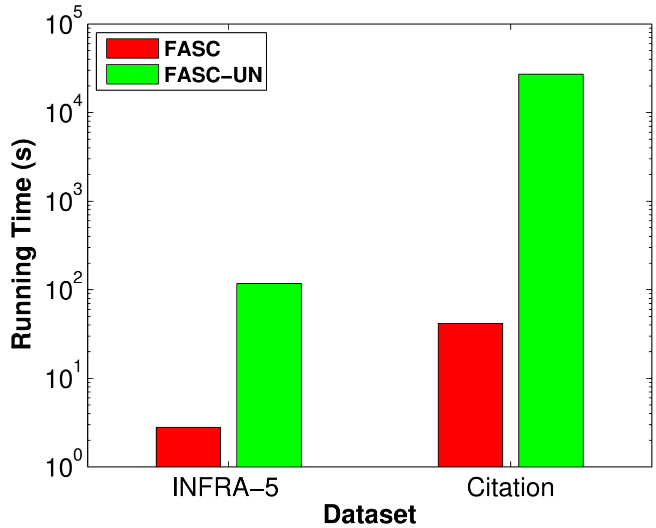


Figure 4.24: Wall-clock Running Time of FASCINATE and FASCINATE-UN.

4.3 Incremental One-Class Collaborative Filtering

The past decade has witnessed the prosperity of recommender systems in various applications, ranging from e-commerce platforms to online service providers. Among the numerous recommendation algorithms in the literature, collaborative filtering based methods are widely adopted in many applications due to its superior effectiveness. Traditional collaborative filtering algorithms are typically designed to provide recommendations based on users' explicit, multi-scale feedback (e.g., rating 1-5). However, in many real applications, the preferences might only be inferred from users' implicit, one-class feedback (e.g., actions or inactions). For example, it is reasonable to infer that a user likes a song if s/he listened to it from the beginning to the end; otherwise, s/he may not be into the song. Such applications are generally formulated as one-class collaborative filtering (OCCF) problems Pan *et al.* (2008).

The key challenges for OCCF lie in the sparsity of positive feedback (preferences) and the ambiguity of missing preferences. A promising way to address those issues is to exploit side information from the social networks of users and/or similarity networks of items as in Ma *et al.* (2011) and Yao *et al.* (2014). The key idea behind those methods is that socially connected users tend to share similar tastes on items; while similar items are more likely to impose similar impact to users.

Most of the existing OCCF algorithms are focused on *static* systems, despite the fact that both user preferences and side networks are evolving over time. For example, in the e-commerce platform shown in Fig. 4.25, new friendship relations (black dashed lines) and user preferences (red dashed lines) are emerging over time. Meanwhile, as new products are being released to the market, similarity links between newly released products and existing products would appear in the system as well. In such a coupled system, the emergence of new connections and preferences may cause a ripple effect on the platform, hence affect the preferences of a large proportion of users. Consequently, it is necessary to update the latent features of users and items obtained in the previous time stamp to accommodate changes. A straightforward way to update the latent features is to rerun the OCCF algorithm from scratch whenever the system changes. However, for large-scale applications, such a strategy may take an unaffordable long time, which would compromise user experience.

In this work, we propose an efficient algorithm to incrementally update one-class collaborative filtering results with co-evolving side networks. To efficiently accommodate the changes in the system, we propose to model the evolution of latent features based on the following observations: the system often evolves smoothly between two consecutive time stamps such that a large number of observed preferences and network links remain unchanged. Thus, we can view the new latent features as a subtle

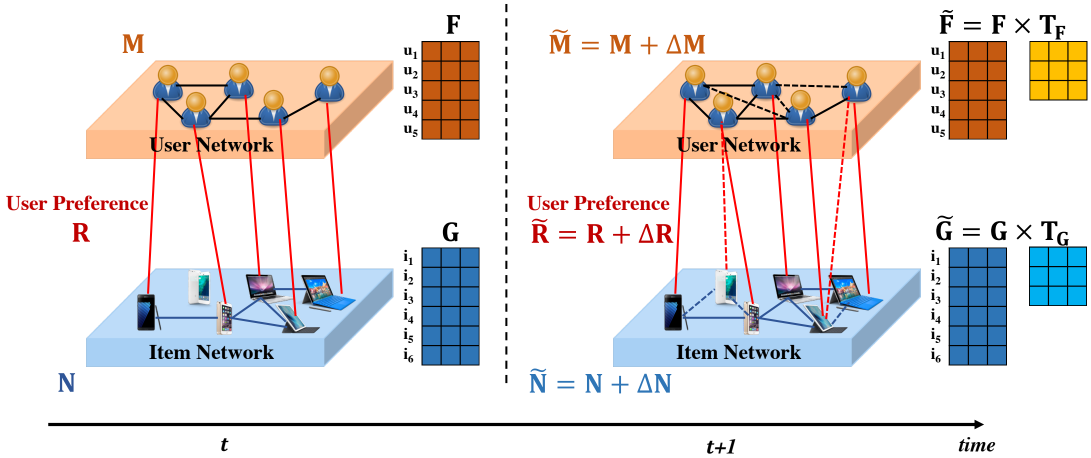


Figure 4.25: An Illustration of Online One-class Recommendation Problem with Side Networks. Solid Lines Represent the Links in the Original System, Dashed Lines Represent the Newly Emerged Links. (Best Viewed in Color.)

linear transformation from the previous features. This would in turn allows us to incrementally solve the OCCF problem in a timely manner without re-solving it from scratch.

The main contributions of this work can be summarized as follows:

- *Problem Formulation.* We formally define the problem of incremental OCCF with co-evolving side networks.
- *Algorithms and Analysis.* We propose an incremental OCCF algorithm (i.e., ENCORE) that can efficiently accommodate system dynamics, and analyze its optimality and complexity.
- *Evaluations.* We empirically evaluate the proposed method on real-world datasets to verify its effectiveness and efficiency.

4.3.1 Problem Definition

In this section, we first give a formal definition of the studied problem of incremental OCCF with co-evolving side networks. After that, we provide the preliminaries to facilitate the understanding of the proposed algorithm.

The main symbols used in this work are summarized in Table 5.3. We use bold uppercase for matrices (e.g., $\mathbf{A}$) and $\Delta\mathbf{A}$ for the perturbation matrix of $\mathbf{A}$. $\tilde{}$ sign denotes the notations after adding the perturbations into the system (i.e., $\tilde{\mathbf{A}} = \mathbf{A} + \Delta\mathbf{A}$). $'$ sign denotes the matrix transpose.

With the above notations, we first define the static OCCF problem with side networks as follows.

Definition 2. *The problem of static OCCF problem with side networks.*

Given: $\Gamma =< \mathbf{M}, \mathbf{N}, \mathbf{R} >$ *where* $\mathbf{M}$ *is an* $n_u \times n_u$ *social network between users;* $\mathbf{N}$ *is an* $n_i \times n_i$ *similarity network between items; and* $\mathbf{R}$ *is an* $n_u \times n_i$ *user preference matrix, in which* $\mathbf{R}(i,j) = 1$ *if user* i *shows preference on item* j*, otherwise* $\mathbf{R}(i,j) = 0$.

Output: *The inferred preference between user* u *and item* i*.*

Based on the above definition, we give the formal definition of incremental OCCF problem with co-evolving side networks.

Problem 4. *The problem of incremental OCCF with co-evolving side networks.*

Given: *(1) The original system* $\Gamma =< \mathbf{M}, \mathbf{N}, \mathbf{R} >$*; (2) the perturbation of the system* $\Delta\Gamma =< \Delta\mathbf{M}, \Delta\mathbf{N}, \Delta\mathbf{R} >$*; (3) the* $n_u \times r$ *latent feature matrix* $\mathbf{F}$ *for users in the original system* Γ*; and (4) the* $n_i \times r$ *latent feature matrix* $\mathbf{G}$ *for items in the original system* Γ*.*

Output: *The inferred preference between user* u *and item* i *in* $\tilde{\Gamma} =< \tilde{\mathbf{M}}, \tilde{\mathbf{N}}, \tilde{\mathbf{R}} >$*.*

Preliminaries

Under static settings, OCCF problem with side networks can be solved with the following optimization problem Yao *et al.* (2014)

Table 4.10: Main Symbols.

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}$	adjacency matrices
$\Delta\mathbf{A}$	perturbation matrix of $\mathbf{A}$
$\tilde{\mathbf{A}}$	updated matrix of $\mathbf{A}$
$\mathbf{A}(i,j)$	the element at i^{th} row j^{th} column in $\mathbf{A}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\mathbf{M}$	the adjacency matrix of user network
$\mathbf{N}$	the adjacency matrix of item network
$\mathbf{D}_M, \mathbf{D}_N$	the diagonal degree matrices for $\mathbf{M}$ and $\mathbf{N}$
$\mathbf{R}$	the preference matrix for users w.r.t. items
Γ	the recommendation problem with side networks $\Gamma =< \mathbf{M}, \mathbf{N}, \mathbf{R} >$
$\mathbf{W}$	the weight matrix for $\mathbf{R}$
$\mathbf{F}$	the latent feature matrix for users
$\mathbf{G}$	the latent feature matrix for items
n_u, n_i	number of users and items
m_u, m_i	number of edges in $\mathbf{M}$ and $\mathbf{N}$
m_r	number of observed links in $\mathbf{R}$
r	the rank for $\mathbf{F}$ and $\mathbf{G}$
t	the number of iterations

$$\min_{\mathbf{F},\mathbf{G}\geq 0} \underbrace{\|\mathbf{W}\odot(\mathbf{R}-\mathbf{F}\mathbf{G}')\|_F^2}_{\text{Matching Observed Ratings}} + \underbrace{\beta(\|\mathbf{F}\|_F^2+\|\mathbf{G}\|_F^2)}_{\text{Regularization}} \\ + \underbrace{\alpha(\mathrm{tr}(\mathbf{F}'(\mathbf{D}_M-\mathbf{M})\mathbf{F})+\mathrm{tr}(\mathbf{G}'(\mathbf{D}_N-\mathbf{N})\mathbf{G}))}_{\text{Node Homophily}} \tag{4.52}$$

where $\odot$ is the Hadamard product with $[\mathbf{A} \odot \mathbf{B}](i,j) = \mathbf{A}(i,j)\mathbf{B}(i,j)$. In the above objective function, $\mathbf{R}$ is the user preference matrix; $\mathbf{F}$ and $\mathbf{G}$ are the low-rank latent feature matrices for users and items, respectively; $\mathbf{D}_M$ and $\mathbf{D}_N$ are the diagonal degree matrices for user network $\mathbf{M}$ and item network $\mathbf{N}$ (i.e., $\mathbf{D}_M(u,u) = \sum_k^{n_u} \mathbf{A}(u,k)$, $\mathbf{D}_N(i,i) = \sum_k^{n_i} \mathbf{A}(i,k)$), respectively. $\mathbf{W}$ is an $n_u \times n_i$ weighting matrix, in which $\mathbf{W}(i,j) = 1$ if $\mathbf{R}(i,j) = 1$ (i.e., positive preference observed between user i and item j), otherwise $\mathbf{W}(i,j) \in (0,1)$ if $\mathbf{R}(i,j) = 0$ (i.e., no preference observed between user i and item j). It is worth to mention that the weight for unobserved links is used to mitigate its uncertainty between potential positive preferences and negative examples. Consequently, different weighting strategies can be applied in different scenarios. In this work, we set the weight of all unobserved entries to a global value w for the ease of computation.

In Eq. (4.52), the first term is used to match the preferences in matrix $\mathbf{R}$; the second term is to prevent overfitting of the model; and the third term is used to exploit node homophily in side networks. The intuition behind this term is that similar users would hold similar preferences to items (i.e., small $\|\mathbf{F}(u,:) - \mathbf{F}(v,:)\|_2^2$). Correspondingly, similar items would possess similar attractiveness to users (i.e., small $\|\mathbf{G}(i,:) - \mathbf{G}(j,:)\|_2^2$). The entire optimization problem in Eq. (4.52) can be solved by non-negative matrix factorization techniques Lee and Seung (2001) with time complexity $O(((m_u + m_i + m_r)r + (n_u + n_i)r^2)t)$ (t is the number of iterations). The inferred preference between user u and item i can be estimated by $\mathbf{F}(u,:)\mathbf{G}(i,:)'$, where $\mathbf{F}$ and $\mathbf{G}$ are the local optimal solutions of Eq. (4.52).

4.3.2 Proposed Algorithm

In this section, we first introduce the proposed algorithm for incremental OCCF with co-evolving side networks. Then we analyze its effectiveness and efficiency.

The proposed Algorithm

Given a static recommendation input $\Gamma =< \mathbf{M}, \mathbf{N}, \mathbf{R} >$, we can find its low-rank feature matrices $\mathbf{F}$ and $\mathbf{G}$ by solving Eq. (4.52) as shown in the previous section. However, in real applications, networks are evolving over time with perturbation $\Delta\Gamma =< \Delta\mathbf{M}, \Delta\mathbf{N}, \Delta\mathbf{R} >$ from the previous time stamp. Consequently, the low-rank feature matrices should be updated accordingly to provide a more accurate preference estimation. In real applications, systems are often changing smoothly, hence we can assume that the updated user features $\tilde{\mathbf{F}}$ and item features $\tilde{\mathbf{G}}$ still reside in the same feature space with $\mathbf{F}$ and $\mathbf{G}$, but are subtly transformed by the system perturbations. In this way, the updated feature matrices $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{G}}$ can be viewed as a linear transformation from $\mathbf{F}$ and $\mathbf{G}$ as shown in the example in Fig. 4.25 (i.e., $\tilde{\mathbf{F}} = \mathbf{F}\mathbf{T}_F$, $\tilde{\mathbf{G}} = \mathbf{G}\mathbf{T}_G$). Therefore, Problem 4 is equivalent to finding the transformation matrices $\mathbf{T}_F$ and $\mathbf{T}_G$ for the new time stamp. Hence, the new objective function under perturbation $\Delta\Gamma$ can be written as

$$
\begin{aligned}
&\min_{\mathbf{T}_F,\mathbf{T}_G} \|\tilde{\mathbf{W}} \odot (\tilde{\mathbf{R}} - \mathbf{F}\mathbf{T}_F\mathbf{T}'_G\mathbf{G}')\|_F^2 \\
&+ \alpha(\mathrm{tr}\mathbf{T}'_F\mathbf{F}'(\mathbf{D}_{\tilde{M}} - \tilde{\mathbf{M}})\mathbf{F}\mathbf{T}_F) \\
&+ \alpha\mathrm{tr}(\mathbf{T}'_G\mathbf{G}'(\mathbf{D}_{\tilde{N}} - \tilde{\mathbf{N}})\mathbf{G}\mathbf{T}_G) + \beta(\|\mathbf{F}\mathbf{T}_F\|_F^2 + \|\mathbf{G}\mathbf{T}_G\|_F^2) \\
&\quad \text{s.t.} \quad \mathbf{F}\mathbf{T}_F, \mathbf{G}\mathbf{T}_G \geq 0
\end{aligned}
\tag{4.53}
$$

Notice that the above objective function imposes a linear constraint on $\mathbf{T}_F$ and $\mathbf{T}_G$ in $\mathbf{F}\mathbf{T}_F, \mathbf{G}\mathbf{T}_G \geq 0$, which would inevitably increase the computational complexity. We propose to simplify the constraint by replacing it with a non-negative constraint on $\mathbf{T}_F$ and $\mathbf{T}_G$. As $\mathbf{F}$ and $\mathbf{G}$ are non-negative in the first place, their non-negative linear combinations $\mathbf{F}\mathbf{T}_F$ and $\mathbf{G}\mathbf{T}_G$ are guaranteed to be non-negative as well. Therefore, we can rewrite the above objective function as follows

$$\begin{aligned}
&\min_{\mathbf{T}_F,\mathbf{T}_G\geq 0} \|\tilde{\mathbf{W}} \odot (\tilde{\mathbf{R}} - \mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}')\|_F^2 \\
&+ \alpha \mathrm{tr}(\mathbf{T}_F'\mathbf{F}'(\mathbf{D}_{\tilde{M}} - \tilde{\mathbf{M}})\mathbf{F}\mathbf{T}_F) \\
&+ \alpha \mathrm{tr}(\mathbf{T}_G'\mathbf{G}'(\mathbf{D}_{\tilde{N}} - \tilde{\mathbf{N}})\mathbf{G}\mathbf{T}_G) + \beta(\|\mathbf{T}_F\|_F^2 + \|\mathbf{T}_G\|_F^2)
\end{aligned} \tag{4.54}$$

As the objective function in Eq. (4.54) is not jointly convex w.r.t. $\mathbf{T}_F$ and $\mathbf{T}_G$ due to the term $\mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}'$, it is hard to find the global optimal solution for the problem. Instead, we seek to obtain its local optimal solution by alternatively updating $\mathbf{T}_F$ and $\mathbf{T}_G$ while fixing the other one.

When $\mathbf{T}_G$ is fixed, the objective function w.r.t. $\mathbf{T}_F$ is reduced to

$$\begin{aligned}
J_{\mathbf{T}_F} =& \|\tilde{\mathbf{W}} \odot (\tilde{\mathbf{R}} - \mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}')\|_F^2 \\
&+ \alpha \mathrm{tr}(\mathbf{T}_F'\mathbf{F}'(\mathbf{D}_{\tilde{M}} - \tilde{\mathbf{M}})\mathbf{F}\mathbf{T}_F) + \beta\|\mathbf{T}_F\|_F^2
\end{aligned} \tag{4.55}$$

Then the derivative of $J_{\mathbf{T}_F}$ w.r.t. $\mathbf{T}_F$ is

$$\begin{aligned}
&\frac{1}{2}\frac{\partial J_{\mathbf{T}_F}}{\partial \mathbf{T}_F} = \mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot (\mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}'))\mathbf{G}\mathbf{T}_G \\
&- \mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot \tilde{\mathbf{R}})\mathbf{G}\mathbf{T}_G + \alpha\mathbf{F}'(\tilde{\mathbf{D}}_{\tilde{\mathbf{M}}} - \tilde{\mathbf{M}})\mathbf{F}\mathbf{T}_F + \beta\mathbf{T}_F
\end{aligned} \tag{4.56}$$

Therefore, we can update $\mathbf{T}_F$ with

$$\mathbf{T}_F(i,j) = \mathbf{T}_F(i,j)\sqrt{\frac{\mathbf{X}_F(i,j)}{\mathbf{Y}_F(i,j)}} \tag{4.57}$$

where

$$\mathbf{X}_F = \mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot \tilde{\mathbf{R}})\mathbf{G}\mathbf{T}_G + \alpha\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F}\mathbf{T}_F \tag{4.58}$$

$$\begin{aligned}
\mathbf{Y}_F =& \mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot (\mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}'))\mathbf{G}\mathbf{T}_G \\
&+ \alpha\mathbf{F}'\tilde{\mathbf{D}}_{\tilde{\mathbf{M}}}\mathbf{F}\mathbf{T}_F + \beta\mathbf{T}_F
\end{aligned} \tag{4.59}$$

Note that the brute force way to update $\mathbf{Y}_F$ requires to calculate a large dense matrix $\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot (\mathbf{F}\mathbf{T}_F\mathbf{T}'_G\mathbf{G}')$ (i.e., $\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot (\tilde{\mathbf{F}}\tilde{\mathbf{G}}')$). This step will take $O(n_u n_i r)$ which is time-consuming in large-scale systems. Recall that we have set $\tilde{\mathbf{W}}(i,j) = 1$ if $\tilde{\mathbf{R}}(i,j) = 1$ and $\tilde{\mathbf{W}}(i,j) = w$ if $\tilde{\mathbf{R}}(i,j) = 0$, then the above term can be rewritten as $(1-w^2)\tilde{\mathbf{R}}_e + w^2\tilde{\mathbf{F}}\tilde{\mathbf{G}}'$ where $\tilde{\mathbf{R}}_e = \tilde{\mathbf{R}} \odot (\tilde{\mathbf{F}}\tilde{\mathbf{G}})$. In other words, the entries in $\tilde{\mathbf{R}}_e$ are the reconstructed preferences of observed links in the updated preference matrix $\tilde{\mathbf{R}}$, which is very sparse in real applications. Moreover, the term $\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot \tilde{\mathbf{R}}$ in Eq. (4.58) is equivalent to $\tilde{\mathbf{R}}$ itself. Therefore, the updating rule for $\mathbf{T}_F$ can be simplified as

$$\mathbf{X}_F = (\mathbf{F}'\tilde{\mathbf{R}}\mathbf{G})\mathbf{T}_G + \alpha(\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F})\mathbf{T}_F \tag{4.60}$$

$$\begin{aligned}\mathbf{Y}_F = &(1-w^2)\mathbf{F}'\tilde{\mathbf{R}}_e\mathbf{G}\mathbf{T}_G + w^2(\mathbf{F}'\mathbf{F})\mathbf{T}_F\mathbf{T}'_G(\mathbf{G}'\mathbf{G})\mathbf{T}_G \\ &+ \alpha(\mathbf{F}'\tilde{\mathbf{D}}_{\tilde{\mathbf{M}}}\mathbf{F})\mathbf{T}_F + \beta\mathbf{T}_F\end{aligned} \tag{4.61}$$

Similarly, $\mathbf{T}_G$ can be updated with

$$\mathbf{T}_G(i,j) = \mathbf{T}_G(i,j)\sqrt{\frac{\mathbf{X}_G(i,j)}{\mathbf{Y}_G(i,j)}} \tag{4.62}$$

where

$$\mathbf{X}_G = (\mathbf{G}'\tilde{\mathbf{R}}'\mathbf{F})\mathbf{T}_F + \alpha(\mathbf{G}'\tilde{\mathbf{N}}\mathbf{G})\mathbf{T}_G \tag{4.63}$$

$$\begin{aligned}\mathbf{Y}_G = &(1-w^2)\mathbf{G}'\tilde{\mathbf{R}}'_e\mathbf{F}\mathbf{T}_F + w^2(\mathbf{G}'\mathbf{G})\mathbf{T}_G\mathbf{T}'_F(\mathbf{F}'\mathbf{F})\mathbf{T}_F \\ &+ \alpha(\mathbf{G}'\tilde{\mathbf{D}}_{\tilde{\mathbf{N}}}\mathbf{G})\mathbf{T}_G + \beta\mathbf{T}_G\end{aligned} \tag{4.64}$$

The proposed algorithm is summarized in Alg. 7. It first gets r, the dimension of latent features $\mathbf{F}$ and $\mathbf{G}$ in step 1, and then initializes the transformation matrices $\mathbf{T}_F$ and $\mathbf{T}_G$ randomly in step 2 and 3. From step 4, the algorithm begins to update $\mathbf{T}_F$ (step 5) and $\mathbf{T}_G$ (step 6) alternatively until convergence.

Algorithm 7 ENCORE: The Incremental OCCF Algorithm with Co-Evolving Side Networks

Input: (1) the original recommendation input $\Gamma =< \mathbf{M}, \mathbf{N}, \mathbf{R} >$; (2) the perturbations on the system $\Delta\Gamma =< \Delta\mathbf{M}, \Delta\mathbf{N}, \Delta\mathbf{R} >$; (3) the original latent features $\mathbf{F}$ and $\mathbf{G}$; (4) weight w; and (5) regularized parameters α and β;

Output: (1) Transformation matrix for user latent features $\mathbf{T}_F$ and (2) transformation matrix for item latent features $\mathbf{T}_G$

1: $r \leftarrow$ rank of $\mathbf{F}$ and $\mathbf{G}$
2: initialize $\mathbf{T}_F$ as $r \times r$ non-negative random matrix
3: initialize $\mathbf{T}_G$ as $r \times r$ non-negative random matrix
4: **while** not converge **do**
5:     update $\mathbf{T}_F$ as Eq. (4.57)
6:     update $\mathbf{T}_G$ as Eq. (4.62)
7: **end while**
8: return $\mathbf{T}_F$, $\mathbf{T}_G$

Algorithm Analysis

We analyze the effectiveness and efficiency of Alg. 7. In terms of the effectiveness of the algorithm, we first show that the fixed point solutions of Eq. (4.57) and Eq. (4.62) satisfy the KKT condition.

Theorem 2. *The fixed point solutions of Eq.* (4.57) *and Eq.* (4.62) *satisfy the KKT condition.*

Proof. As $\mathbf{T}_F$ and $\mathbf{T}_G$ are solved in the same way, we only need to show that the fixed point solution for $\mathbf{T}_F$ in Eq. (4.57) satisfy the KKT condition, the other one can be proved in the same procedure.

First, the Lagrangian function for Eq. (4.55) is

$$\begin{aligned} L_{J_F} =& \|\tilde{\mathbf{W}} \odot (\tilde{\mathbf{R}} - \mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}')\|_F^2 + \mathrm{tr}(\mathbf{T}_F'\mathbf{F}'\mathbf{D}_{\tilde{M}}\mathbf{F}\mathbf{T}_F) \\ &- \alpha\mathrm{tr}(\mathbf{T}_F'\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F}\mathbf{T}_F) + \beta\|\mathbf{T}_F\|_F^2 - \mathrm{tr}(\Lambda'\mathbf{T}_F) \end{aligned} \tag{4.65}$$

where Λ is the Lagrange multiplier. By setting the derivative of L_{J_F} w.r.t. $\mathbf{T}_F$ to 0, we get

$$\begin{aligned} &2(\mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot (\mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}'))\mathbf{G}\mathbf{T}_G \\ &- \mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot \tilde{\mathbf{R}})\mathbf{G}\mathbf{T}_G \\ &+ \alpha\mathbf{T}_F\mathbf{F}'\tilde{\mathbf{D}}_{\tilde{M}}\mathbf{F} - \alpha\mathbf{T}_F\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F} + \beta\mathbf{T}_F) = \Lambda \end{aligned} \tag{4.66}$$

By the KKT slackness condition, we have

$$\begin{aligned} &[-\mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot \tilde{\mathbf{R}})\mathbf{G}\mathbf{T}_G - \alpha\mathbf{T}_F\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F} \\ &+ \mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot (\mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}'))\mathbf{G}\mathbf{T}_G \\ &+ \alpha\mathbf{T}_F\mathbf{F}'\tilde{\mathbf{D}}_{\tilde{M}}\mathbf{F} + \beta\mathbf{T}_F](i,j)\mathbf{T}_F(i,j) = 0 \end{aligned} \tag{4.67}$$

At the fixed point of Eq. (4.57), we have $\mathbf{X}_F(i,j) = \mathbf{Y}_F(i,j)$, which implies that

$$\begin{aligned} &[\mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot \tilde{\mathbf{R}})\mathbf{G}\mathbf{T}_G + \alpha\mathbf{T}_F\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F}](i,j) \\ &= [\mathbf{F}'(\tilde{\mathbf{W}} \odot \tilde{\mathbf{W}} \odot (\mathbf{F}\mathbf{T}_F\mathbf{T}_G'\mathbf{G}'))\mathbf{G}\mathbf{T}_G \\ &\quad + \alpha\mathbf{T}_F\mathbf{F}'\tilde{\mathbf{D}}_{\tilde{M}}\mathbf{F} + \beta\mathbf{T}_F](i,j) \end{aligned} \tag{4.68}$$

Clearly, the above equation satisfies the KKT slackness condition in Eq. (4.67). Therefore, the fixed point solution of Eq. (4.57) satisfies the KKT condition. □

Theorem 2 states that the updating rules in Eq. (4.57) lead to a local optimal solution to Eq. (4.55) at convergence. Also, it can be proved that by following the updating rule in Eq. (4.57), the objective function in Eq. (4.55) decreases monotonically.

Combining Theorem 2 with the monotonic decreasing property, we can conclude that Alg. 7 converges to the local optimal solution $\mathbf{T}_F$ for the objective function in Eq. (4.55). Similarly, we have the local optimal solution $\mathbf{T}_G$. The two matrices $\mathbf{T}_F$ and $\mathbf{T}_G$ form the local optimal solution for Eq. (4.54).

For efficiency of the algorithm, we analyze the time complexity and space complexity of Alg. 7 in Lemma 4.3.2 and Lemma 4.3.2 respectively. The time complexity of proposed algorithm is $O((\tilde{m}_u + \tilde{m}_i)r + ((n_u + n_i + r)r^2 + \tilde{m}_r r)t)$.

Proof. In Alg. 7, as term $\tilde{\mathbf{M}}$, $\tilde{\mathbf{N}}$, $\tilde{\mathbf{R}}$, $\mathbf{F}$, and $\mathbf{G}$ remain the same during the iterations, we can pre-compute related constant terms to avoid redundant computations. The complexities of computing constant terms in Eq. (4.60) - (4.64) are $O(n_i r^2 + \tilde{m}_r r)$ for $\mathbf{F}'\tilde{\mathbf{R}}\mathbf{G}$; $O(n_u r^2 + \tilde{m}_u r)$ for $\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F}$; $O(n_u r^2)$ for $\mathbf{F}'\mathbf{D}_{\tilde{M}}\mathbf{F}$; $O(n_u r^2)$ for $\mathbf{F}'\mathbf{F}$; $O(n_i r^2)$ for $\mathbf{G}'\mathbf{G}$; $O(n_i r^2 + \tilde{m}_i r)$ for $\mathbf{G}'\tilde{\mathbf{N}}\mathbf{G}$; and $O(n_i r^2)$ for $\mathbf{G}'\mathbf{D}_{\tilde{N}}\mathbf{G}$. Thus, the complexity for pre-computing is $O((n_u + n_i)r^2 + (\tilde{m}_u + \tilde{m}_i + \tilde{m}_r)r)$. In each iteration, it takes $O(n_u r^2)$ and $O(n_i r^2)$ to compute $\mathbf{F}\mathbf{T}_F$ (i.e., $\tilde{\mathbf{F}}$) and $\mathbf{G}\mathbf{T}_G$ (i.e., $\tilde{\mathbf{G}}$) respectively. The complexity of computing $\mathbf{F}'\tilde{\mathbf{R}}_e\mathbf{G}\mathbf{T}_G$ is $O(n_i r^2 + \tilde{m}_r r)$, the rest of the computations for updating $\mathbf{T}_F$ and $\mathbf{T}_G$ are both $O(r^3)$. Therefore, the overall complexity for Alg. 7 is $O((\tilde{m}_u + \tilde{m}_i)r + ((n_u + n_i + r)r^2 + \tilde{m}_r r)t)$, where t is the number of iterations in the algorithm. □

Compared with the complexity of static OCCF algorithm in the previous section ($O(((m_u + m_i + m_r)r + (n_u + n_i)r^2)t)$), the proposed ENCORE is more efficient, with an $O((\tilde{m}_u + \tilde{m}_i)r)$ reduction in the time complexity in *each* iteration.

The space complexity of proposed algorithm is $O((n_u + n_i + r)r + \tilde{m}_u + \tilde{m}_i + \tilde{m}_r)$.

Proof. The algorithm requires a space of $O(n_u r + n_i r)$ to store $\mathbf{F}$ and $\mathbf{G}$, $O(r^2)$ to store transformation matrices $\mathbf{T}_F$ and $\mathbf{T}_G$, and $O(\tilde{m}_u + \tilde{m}_i + \tilde{m}_r)$ to store the updated rating matrix and side networks. The space needed to compute and store the constant terms are $O(n_i r+r^2)$ for $\mathbf{F}'\tilde{\mathbf{R}}\mathbf{G}$; $O(n_u r+r^2)$ for $\mathbf{F}'\tilde{\mathbf{M}}\mathbf{F}$ and $\mathbf{F}'\mathbf{D}_{\tilde{M}}\mathbf{F}$; $O(r^2)$ for $\mathbf{F}'\mathbf{F}$ and $\mathbf{G}'\mathbf{G}$; $O(n_i r + r^2)$ for $\mathbf{G}'\tilde{\mathbf{N}}\mathbf{G}$ and $\mathbf{G}'\mathbf{D}_{\tilde{N}}\mathbf{G}$. Therefore, the space costs for computing constant terms is $O((n_u+n_i)r+r^2)$. In each iteration, it takes a space of $O((n_u+n_i)r)$ to compute $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{G}}$, $O(\tilde{m}_r)$ to compute $\tilde{\mathbf{R}}_e$, $O(n_i r+r^2)$ to compute $\mathbf{F}'\tilde{\mathbf{R}}_e\mathbf{G}\mathbf{T}_G$, $O(r^2)$ for the rest of the matrix multiplications to update $\mathbf{T}_F$ and $\mathbf{T}_G$. Putting all together, the overall space complexity for Alg. 7 is $O((n_u + n_i + r)r + \tilde{m}_u + \tilde{m}_i + \tilde{m}_r)$. $\square$

Variations

In this section, we discuss some of the variants of ENCORE. First, when the weighting matrix $\mathbf{W}$ is an all-one matrix, ENCORE becomes a dynamic clustering algorithm, in which $\mathbf{T}_F$ and $\mathbf{T}_G$ can be viewed as the cluster membership transition matrix. Second, when one or both sides networks are missing, the corresponding regularization term would be removed from the objective function. In particular, when both side of the networks were removed, ENCORE is reduced to a dynamic algorithm for classic one class collaborative filtering problem.

4.3.3 Experimental Evaluations

In this section, we evaluate the proposed ENCORE algorithm on two real datasets. The experiments are designed to answer the following two questions.

- *Effectiveness*. How effective is ENCORE for OCCF problem with co-evolving side networks?
- *Efficiency*. How fast is ENCORE compared with batch-mode static counterpart?

Experimental Setup

We first introduce the datasets used, comparing methods, evaluation metrics, and experimental settings before presenting the details of the experiments.

Datasets Description. We use two real datasets Ciao Tang *et al.* (2012a) and Epinions Tang *et al.* (2012b) to evaluate the proposed ENCORE method. Ciao and Epinions are two popular online product review websites in which users are allowed to build connections and share experiences on the products with each other. To fit the one-class collaborative filtering problem, all missing links and ratings that are no greater than 3 are viewed as negative examples (i.e., labeled as 0); while ratings that are greater or equal to 4 are marked as positive examples (i.e., labeled as 1). The user side network contains the trust relations between users, while the item side network describes the similarity between items based on their reviews [6] . Both datasets have been preprocessed and used in Yao *et al.* (2014) and are publicly available. The statistics of the datasets are summarized in Table 4.11.

In our evaluation, the datasets are partitioned into three groups. The first group is the original training system which contains 50% of the ratings and the corresponding side network links; the second group is the incremental system, which adds 1% links to rating matrix and the side network connections at each time stamp; the last group is the testing system, which contains the rest of the data.

Comparing Methods. We compare ENCORE with the following baseline methods to demonstrate its effectiveness.

- **ReRun**. *ReRun* is the batch-mode static counterpart for ENCORE. At each time stamp, it takes the current system snapshot as input networks and solves the optimization problem in Eq. (4.52) from scratch. As *ReRun* does not impose

[6]Similarity between items is calculated by the cosine similarity between TF-IDF word vectors constructed from item reviews.

Table 4.11: Statistics of Datasets.

Dataset	Ciao	Epinions
# of users	6,102	33,725
# of items	12,082	43,542
# of user links	151,722	656,910
# of items links	283,284	498,794
# of preferences	117,731	500,478

any transformational constraints on the latent features in two consecutive time stamps, it can be used to validate the effectiveness of the transformation model in ENCORE.

- **M+R**. *M+R* is a variant of ENCORE, which only contains user side network and preference matrix in the system.
- **N+R**. *N+R* is another variant of ENCORE, which only contains item side network and preference matrix.
- **R-MF**. *R-MF* is a simple method for OCCF proposed in Pan *et al.* (2008), which only utilizes the preference matrix in the system for the recommendation.
- **R-SGD**. *R-SGD* shares the same objective function with *R-MF*. Instead of calculating the latent features at each time stamp, *R-SGD* modifies related latent features with newly emerged ratings by stochastic gradient descent method.

Evaluation Metrics. In our experiments, we assess the effectiveness of ENCORE with *MAP* and *R-MPR* as evaluation metrics.

- **MAP**. *MAP* (Mean Average Precision) is originally used to evaluate ranked documents over a set of queries. Here it computes the mean average precision over all users in the test set Pan *et al.* (2008). The larger the *MAP* is, the better the performance is.

- **R-MPR**. *R-MPR* (Reverse Mean Percentage Ranking) is a variation of *MPR*, which is originally used to evaluate users' satisfaction of items by a ranked list. A randomly generated item list can achieve an *MPR* of 50% Li *et al.* (2010). The smaller the *MPR* is, the better the performance is. Here we set *R-MPR* to be 0.5-*MPR*, thus a larger *R-MPR* indicates better performance.

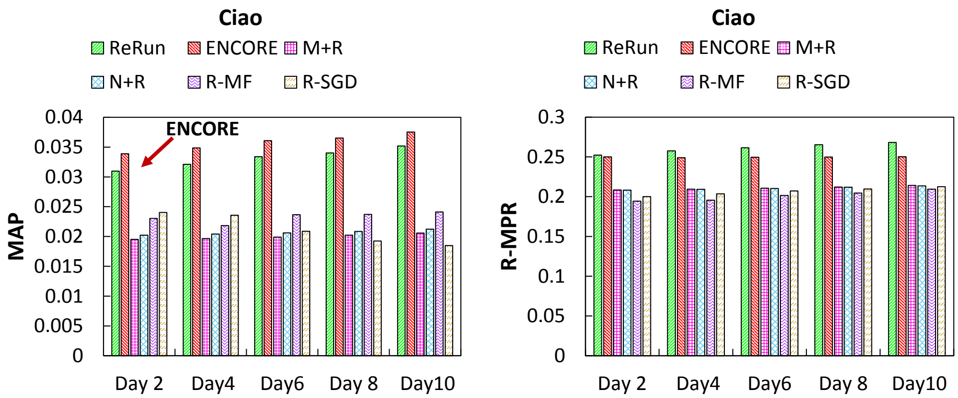
Machine. The experiments are performed on a machine with 2 Intel Xeon 3.5GHz processors and 256GB of RAM. The algorithms are implemented with MATLAB using a single thread.

Effectiveness Results

We compare the proposed algorithm with other methods on both Ciao and Epinions datasets. The results are shown in Fig. 4.26 and Fig. 4.27. We make the following observations from these two figures.

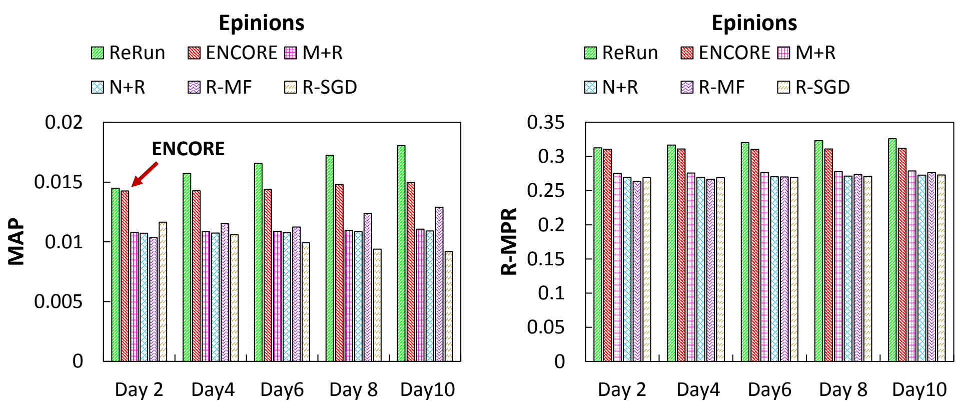
- In both datasets, ENCORE achieves close performance with *ReRun*. Such results demonstrate that ENCORE can effectively accommodate newly emerged links in the dynamic system for the recommendation.

- Side networks between users and items are both important for improving the quality of recommendation results. As we can see from Fig. 4.26 and Fig. 4.27, user network or item network alone with the preference matrix cannot boost the recommendation quality compared with the methods that use preference matrix



(a) MAP (b) R-MPR

Figure 4.26: The Effectiveness Results on Ciao. Higher Is Better. Our Method (Marked by Arrow) Perform Closely with *Rerun* Method. (Best Viewed in Color.)



(a) MAP (b) R-MPR

Figure 4.27: The Effectiveness Results on Epinions. Higher Is Better. Our Method (Marked by Arrow) Performs Closely with *Rerun* Method. (Best Viewed in Color.)

only. However, when both networks are added to the model, the performance can be improved significantly.

Efficiency Results

We evaluate the efficiency of ENCORE on both Ciao and Epinions. As the results are similar, we only report the one on Ciao for brevity. It can be seen from Fig. 4.28

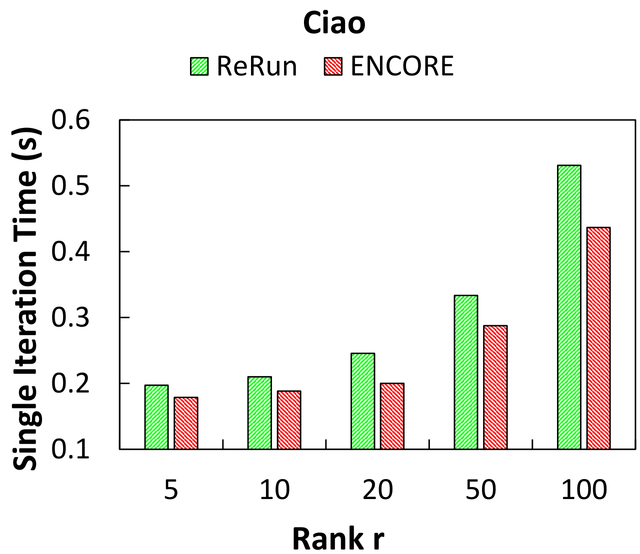


Figure 4.28: The Running Time of *Rerun* Vs. ENCORE for a Single Iteration.

that the average running time of ENCORE for a single iteration is shorter than the *ReRun* method. Specifically, as rank r increases, the speed-up of ENCORE compared to *ReRun* becomes larger accordingly. This observation is consistent with our time complexity analysis in the previous section that the proposed ENCORE algorithm has an $O(r)$ factor speed-up over its static counterpart (*ReRun*). Moreover, as shown in Fig. 4.29, the average running time of ENCORE for one time stamp is much shorter than *ReRun* (with around 75% improvement). This is mainly due to the fact that ENCORE has much fewer variables to optimize at each time stamp compared to *ReRun* (i.e., $2r \times r$ vs. $(m+n) \times r$), which makes it converges faster in a small number of iterations.

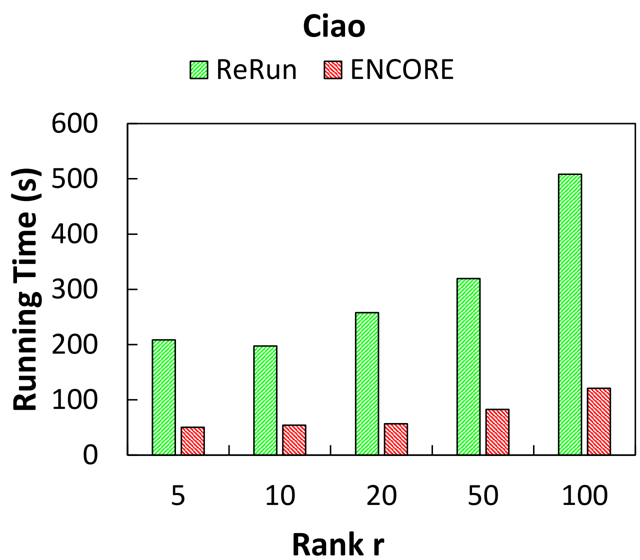


Figure 4.29: The Running Time of *ReRun* Vs. ENCORE for One Time Stamp.

Chapter 5

NETWORK CONNECTIVITY OPTIMIZATION

In this section, we introduce the optimization strategies for SubLine connectivity measures and its extension to multi-layered networks.

5.1 SubLine Connectivity Optimization

As we have discussed in Chapter 3, the connectivity of the network may take different forms depending on the types of the application. Despite the various forms, the connectivity minimization problem has always been a fundamental task in most of the applications, in which a less connected network or subnetwork is more preferred Kovacs and Barabasi (2015). The primary goal for connectivity minimization is to find a set of nodes/edges whose removal may lead to the destruction of the underlying network. For example, in the critical infrastructure construction scenario, the high impact facilities and links identified by the connectivity minimization algorithms can be viewed as the backbone of the network, which is essential to ensure the full functioning of the entire system. While in the immunization scenario, disease control centers need to vaccine high impact entities and cut down highly contagious connections to prevent the prevalence of the disease.

The main computation obstacle for the connectivity minimization problems lies in its combinatorial nature. Specifically, for the global connectivity minimization problem, suppose the number of nodes and edges in the network is n and m respectively, then the number of all possible node sets of size k would be $\binom{n}{k}$ and the number of all possible edge sets is $\binom{m}{k}$. Such exponential complexity would make exhaust search intractable even in mid-sized networks.

To reduce the exponential time complexity, existing algorithms predominantly rely on the greedy scheme. Taking the node deletion based connectivity minimization problem for an example, the greedy scheme would iteratively collect the node that has the largest impact on the pre-defined connectivity in the network until the budget is used up. In virtue of the diminishing returns property on a wide-range of the connectivity minimization problems Chen *et al.* (2015), the greedy scheme can secure a near-optimal approximated solution with an approximation ratio of $1 - 1/e$ Nemhauser *et al.* (1978). A key step in the greedy scheme is to calculate the impact of each candidate node/edge on the given connectivity measure, which often involves eigen-decomposition operations with *polynomial* complexity w.r.t. the size of the network. Obviously, a polynomial algorithm still can not handle large-scale networks efficiently. To further accelerate the algorithm, matrix perturbation based methods are frequently used to approximate the impact of a node/edge Chen *et al.* (2016a). Such approximation algorithms have been proved to scale linearly w.r.t. the network size, while exhibiting empirical superiority over other alternative methods.

Although the above-mentioned methods are empirically effective for some specific connectivity minimization problems, two main challenges for the general connectivity minimization problem still remain largely open. On the theoretical side, the *hardness* of the general connectivity minimization problem has never been systematically justified except for a few special instances (e.g., epidemic threshold Chen *et al.* (2016b) and triangle capacity Li and Yu (2015)). Furthermore, although the greedy scheme can guarantee a $1 - 1/e$ approximation ratio for the connectivity minimization problem, it still remains unknown if a better approximation ratio can be achieved within polynomial time. On the algorithmic side, exact greedy algorithms often bear polynomial time complexity, which is not scalable in large-scale networks. Although matrix perturbation based approximation methods can simplify the complexity down to the linear scale, their optimization quality is highly dependent on the spectrum of the underlying network (the optimization quality would deteriorate quickly in networks with small eigen-gaps Chen and Tong (2017); Le *et al.* (2015)).

In this work, we address the theoretical and algorithmic challenges of the connectivity minimization problem. The main contributions of this work can be summarized as follow.

- *Revealing the Fundamental Limits.* We prove that for the connectivity minimization problem on a wide range of connectivity measures, (1) is NP-complete and (2) $(1-1/e)$ is the best approximation ratio for any polynomial algorithms, unless $NP \subseteq DTIME(n^{O(\log \log n)})$ [1] .

- *Developing New Algorithms.* We propose an effective algorithm (CONTAIN) for network connectivity optimization. The centerpieces of the proposed method include (a) an effective impact score approximation method and (b) an efficient eigen-pair update method. The proposed CONTAIN algorithm bears three distinct advantages over the existing methods, including (1) *effectiveness*, being able to handle small eigen-gap networks, consistently outperforming the state-of-the-art methods over a diverse set of real networks; (2) *scalability*, with a linear complexity w.r.t. the network size; and (3) *generality*, applicable to a variety of different network connectivity measures (e.g., leading eigenvalue, triangle capacity, and natural connectivity) as well as network operations (node vs. edge deletion). In addition, we also propose a variation of CONTAIN (CONTAIN+) which can further simply the computational complexity by deriving a closed-form approximation on node/edge impact scores.

[1] $DTIME(t(n))$: the collection of languages that are decidable by $O(t(n))$ time deterministic Turing machineSipser (1997).

5.1.1 Problem Definition

In this section, we formally introduce the network connectivity optimization problem and review the general strategy of greedy algorithms.

Table 5.3 gives the main symbols used in this work. Following the convention, we use bold upper-case for matrices (e.g., $\mathbf{A}$), bold lower-case for vectors (e.g., $\mathbf{a}$) and calligraphic for sets (e.g., $\mathcal{A}$). We use ˜ to denote the notations after node/edge deletion, and Δ to denote the perturbations (e.g., $\Delta\mathbf{A} = \tilde{\mathbf{A}} - \mathbf{A}$). $C(G)$ [2] represents the network connectivity measure to be optimized in G; o indicates an element (a node/edge) in network G; $I(o)$ denotes the impact score of element o on $C(G)$; $\mathbf{\Lambda}$ and $\mathbf{U}$ denote the eigenvalue matrix and eigenvector matrix for the adjacency matrix $\mathbf{A}$ of the network.

Recall that in Chapter 3, we define the SubLine connectivity as

$$C(G) = \sum_{\pi \in G} f(\pi) \tag{5.1}$$

where π is a subgraph of G, f is a non-negative function that maps any subgraph in G to a non-negative real number (i.e., $f : \pi \to \mathbb{R}^+$). Specifically, we have $f(\phi) = 0$ for empty set ϕ; when $f(\pi) > 0$, we call subgraph π as a *valid subgraph*. In other words, the network connectivity $C(G)$ can be viewed as a weighted aggregation of the connectivities of all valid subgraphs in the network. When the valid subgraphs are defined on a subset of nodes $\mathcal{T}$, the connectivity measure defined in Eq. (5.1) can also be extended to measure the local connectivity of a subset of nodes $\mathcal{T}$, where we define $f(\pi) > 0$ iff π is incident to the node set $\mathcal{T}$. Moreover, by choosing an appropriate $f()$ function (refer to 3 for details), Eq. (5.1) includes several prevalent network connectivity measures, e.g., path capacity, triangle capacity, and natural connectivity.

[2] $C(G)$ is the abbreviation for $C(G, f)$, which is related to both the network structure and connectivity function f.

Table 5.1: Main Symbols for CONTAIN.

Symbol	Definition and Description
$G(V,E)$	an undirected network
$\mathbf{A},\mathbf{B}$	the adjacency matrices (bold upper case)
$\mathbf{a},\mathbf{b}$	column vectors (bold lower case)
$\mathcal{A},\mathcal{B}$	sets (calligraphic)
$\mathbf{A}(i,j)$	the element at the i^{th} row and the j^{th} column in $\mathbf{A}$
$\mathbf{a}(i)$	the i^{th} element of vector $\mathbf{a}$
$\mathbf{A}'$	transpose of matrix $\mathbf{A}$
$\Delta\mathbf{A}$	perturbation of $\mathbf{A}$
$\tilde{\mathbf{A}}$	the adjacency matrix after node/edge deletion on $\mathbf{A}$
m,n	number of edges and nodes in network G
$C(G)$	connectivity measure of network G under mapping function f
$C_{\mathcal{T}}(G)$	the local connectivity of subgraph $\mathcal{T}$ on network G
$F(\mathbf{\Lambda}^{(r)})$	associated eigen-function for $C(G)$
o	a network element in G (a node/edge)
$I(o)$	connectivity impact score of o on $C(G)$
$I_{\mathcal{T}}(o)$	local connectivity impact score of o on $C_{\mathcal{T}}(G)$
$\lambda,\mathbf{u}$	the leading eigenvalue and eigenvector of $\mathbf{A}$ (in magnitude)
$\mathbf{\Lambda},\mathbf{U}$	the eigenvalue and eigenvector matrix of $\mathbf{A}$
$\mathbf{\Lambda}^{(r)},\mathbf{U}^{(r)}$	the top-r eigen-pairs of $\mathbf{A}$ (in magnitude)
k	the budget

In terms of computation, it is often much more efficient to either approximate or compute these connectivity measures by the associated eigen-function $F(\mathbf{\Lambda}^{(r)})$, where $\mathbf{\Lambda}^{(r)}$ represents the top-r eigenvalues of $\mathbf{A}$.

Network Connectivity Minimization

With the network connectivity measure in Eq. (5.1), we formally define the network connectivity optimization problem as follows.

Problem 5. Network Connectivity Minimization (NETCOM)

Given: *(1) a network G; (2) a connectivity mapping function $f : \pi \rightarrow \mathbb{R}^+$ which defines $C(G)$; (3) a type of network operation (node deletion vs. edge deletion) and (4) an integer budget k with $1 < k < \min\{|\mathcal{S}_\pi|, K\}$ where $\mathcal{S}_\pi = \{\pi | f(\pi) > 0\}$ denotes the set of valid subgraphs and K denotes the number of valid network elements.*

Output: *a set of network elements $\mathcal{X}$ of size k, whose removal from G would minimize connectivity $C(G)$.*

It is worth noting that depending on the definition of $C(G)$, the valid subgraphs in $\mathcal{S}_\pi$ may have various structures. In the triangle minimization scenario, $\mathcal{S}_\pi$ contains all the triangles in the network. When the valid subgraph shares the same form as the operation type (i.e., a valid subgraph is a single node in node-level operation scenario, or a valid subgraph is an edge in edge-level operation scenario), we call this kind of valid subgraphs as *singletons*. In Problem 5, we also require that the budget $1 < k < \min\{|\mathcal{S}_\pi|, K\}$. This is a fairly generic constraint which can be easily met. For example, for the node deletion operation, the set of valid network elements is simply the entire node set of the input network (i.e., $K = n$); for a connected network with its connectivity measure $C(G)$ defined as the path capacity, we have that $|\mathcal{S}_\pi| > n$. Therefore, the above constraint simply means that we cannot delete all the nodes from the input network, which would make the problem trivial. On the other end of the spectrum, we require that the budget $k > 1$. Otherwise (with $k = 1$), the problem can be easily solved in polynomial time (e.g., by choosing the valid network element with the largest impact score). Problem 5 provides a general definition of the network connectivity optimization problem, which can be in turn instantiated into different instances, depending on (1) the specific choice of the connectivity measure $C(G)$ (or equivalently the choice of the $f()$ function), and (2) the type of network operation (node deletion vs. edge deletion). For example, in the robustness analysis of the power grid, we might choose the natural connectivity as $C(G)$ to evaluate the robustness of the system, and we are interested in identifying k most critical power transmission lines whose failure would cause a cascading failure of the entire grid. To abstract it as a network connectivity optimization problem, we have the input network set as the topological structure of the power grid; the connectivity to optimize as the natural connectivity; the operation type as edge deletion; and the valid network elements as all the edges (i.e., $K = m$ in this case).

The NETCOM problem can be easily extended to local connectivity measures. Specifically, the local connectivity minimization problem can be defined as follows.

Problem 6. Local Connectivity Minimization

Given: *(1) a network G; (2) a subset of target nodes $\mathcal{T}$; (3) a connectivity mapping function $f : \pi \to \mathbb{R}^+$ which defines the local connectivity measure $C_{\mathcal{T}}(G)$; (3) a type of network operation (node deletion vs. edge deletion) and (4) an integer budget k with $1 < k < \min\{|\mathcal{S}_\pi|, K\}$ where $\mathcal{S}_\pi = \{\pi | f(\pi) > 0\}$ denotes the set of valid subgraphs and K denotes the number of valid network elements.*

Output: *a set of network elements $\mathcal{X}$ of size k with $\mathcal{X} \cap \mathcal{T} = \Phi$, whose removal from G would minimize connectivity $C_{\mathcal{T}}(G)$.*

Note that in Problem 6, the restriction $\mathcal{X} \cap \mathcal{T} = \Phi$ on $\mathcal{X}$ is used to avoid the trivial solution under node deletion operations, in which target nodes $\mathcal{T}$ are removed for local connectivity minimization.

Greedy Strategy for NETCOM

Due to the combinatorial nature of Problem 5, it is computationally infeasible to solve it in a brute-force manner. Thanks to the diminishing returns property of

NETCOM, the greedy strategy has become a prevalent choice for solving Problem 5 with a guaranteed $(1-1/e)$ approximation ratio. For the ease of following discussions, we present the outline of such greedy strategy in Algorithm 8. In Algorithm 8, the solution set $\mathcal{X}$ is initialized with an empty set. At each iteration (step 2 to step 8), the element (a node or an edge) with the highest impact score is added to the solution set $\mathcal{X}$ until the budget is reached. The returned solution set $\mathcal{X}$ in step 9 guarantees a $(1-1/e)$ approximation ratio. For more details and proofs, please refer to Chen *et al.* (2015).

Algorithm 8 A Generic Greedy Strategy for NETCOM Chen *et al.* (2015)

Input: (1) A network G; (2) a connectivity mapping function $f : \pi \to \mathbb{R}^+$ which defines $C(G)$; (3) a type of network operation and (4) a positive integer k

Output: a set of network elements $\mathcal{X}$ of size k.

1: initialize $\mathcal{X}$ to be empty
2: **for** $i = 1$ to k **do**
3:     **for** each valid network element o in G **do**
4:         calculate $I(o) \leftarrow C(G) - C(G \setminus \{o\})$
5:     **end for**
6:     add the element $\tilde{o} = \text{argmax}_o I(o)$ to $\mathcal{X}$
7:     remove the element $\{\tilde{o}\}$ from network G
8: **end for**
9: return $\mathcal{X}$

5.1.2 Fundamental Limits

In this section, we start with detailing the theoretic challenges of the network connectivity optimization (NETCOM) problem, and then reveal two fundamental limits, including its hardness and its approximability.

Theoretic Challenges of NETCOM

The first theoretic challenge of NETCOM lies in its hardness. Since the NETCOM problem has various instances, intuitively, the hardness of those instances might vary dramatically from one to another. For example, if the elements in the valid subgraph set $\mathcal{S}_\pi$ are all singletons w.r.t. the corresponding operation type (i.e., $\mathcal{S}_\pi$ is the node set of the input network for the node-level optimization problem, or $\mathcal{S}_\pi$ is the edge set for the edge-level optimization problem), we can simply choose the top–k nodes/edges with the highest $f(\pi)$ scores, which immediately gives the optimal solution. However, if NETCOM is instantiated as an edge minimization problem under node deletion operations (i.e., the valid subgraph $\mathcal{S}_\pi$ consists of all the edges, the valid network element set is the entire node set) , the problem would become the (weighted) max-k vertex cover problem, which is known to be NP-hard. Such observations naturally give rise to the following question, *what is the key intrinsic property of valid subgraph set $\mathcal{S}_\pi$ in conjunction with the network operation type that determines whether or not the corresponding* NETCOM *instance is polynomially solvable?* To date, the hardness of the general NETCOM problem has largely remained unvalidated, except for a few special instances. The second theoretic challenge of NETCOM lies in its approximability. The greedy algorithm outlined in Section 5.1.1 has a provable $(1 - 1/e)$ approximation ratio Chen *et al.* (2015). However, we still do not know if such an approximation ratio is *optimal*. In other words, it remains unknown if there exists any polynomial algorithm with an approximation ratio better than $(1 - 1/e)$ for NETCOM.

Fundamental Limit #1: NP-Completeness

We reveal the hardness result of the NETCOM problem in Theorem 3. It states that the NETCOM problem defined in Problem 5 is in general NP-complete, unless the valid subgraphs in set $\mathcal{S}_\pi$ are mutually independent to each other [3] .

Theorem 3. NP-Completeness of NETCOM. *The* NETCOM *problem with non-independent valid subgraphs in Problem 5 is NP-complete.*

Proof. As the NETCOM problem admits two possible network operations, including node deletions and edge deletions, we present our proof for each scenario in the following two lemmas. Lemma 7 together with Lemma 8 would prove that NETCOM problem is NP-complete. □

Lemma 7. *The k-node connectivity minimization problem is NP-complete.*

Proof. By Eq. (5.1), the connectivity of network G is defined as $C(G) = \sum_{\pi \in G} f(\pi)$. We set function f as

$$f(\pi) = \begin{cases} w_\pi > 0 & \text{if } \pi \text{ is a valid subgraph} \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

Hence, we formulate the k-node minimization problem as follows.

Problem 7. k-Node Minimization Problem: $NodeMin(G, k)$

Given: *(1) A network $G =< V, E >$; (2) the connectivity function f as defined in Eq.* (5.2)*; and (3) the budget k.*

Output: *A set with k nodes $V' \subseteq V$, such that $C(G \setminus V')$ (i.e., the connectivity in $G \setminus V'$) is minimized.*

[3]Two valid subgraphs are independent to each other if they do not have any common valid network element.

Here we first prove that $NodeMin(G, k)$ is NP-hard by constructing a polynomial reduction from a well-known NP-hard problem, the max k-coverage problem (i.e., $MaxCover(n, m, k)$) Kleinberg and Tardos (2006). The $MaxCover(n, m, k)$ problem is defined as follows.

Problem 8. Max k-Coverage Problem: $MaxCover(n, m, k)$

Given: *(1) the universal set of n elements $\mathcal{U} = \{e_1, e_2, \ldots, e_n\}$; (2) a collection $\mathcal{S} = \{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ of m distinct subsets of $\mathcal{U}$, which are not mutually exclusive; (3) the non-negative weights $\mathcal{W} = \{w_i, \ldots, w_n\}$ associated to the corresponding elements in $\mathcal{U}$; and (4) a positive integer k.*

Output: *A set $\mathcal{S}' \subseteq \mathcal{S}$ with $|\mathcal{S}'| \leq k$, s.t. $\sum_{e_i \in \mathcal{U}'} w_i$ is maximized, where $\mathcal{U}'$ is the set of elements covered by the sets in $\mathcal{S}'$.*

We aim to prove that $MaxCover(n, m, k)$ is polynomially reducible to the k-node minimization problem $NodeMin(G, k)$ (i.e., $MaxCover(n, m, k) \leq_p NodeMin(G, k)$). Without loss of generality, we assume that $1 < k < m$. The rationality behind this assumption is that when $k = 1$, $MaxCover(n, m, 1)$ can be trivially solved by picking the set in $\mathcal{S}$ that contains the elements with the largest weight sum (i.e., S'=$\{\arg\max_{\mathcal{B} \in \mathcal{S}} \sum_{e_i \in \mathcal{B}} w_i\}$); while for $k \geq m$, we may just take all the subsets in $\mathcal{S}$ into $\mathcal{S}$' to guarantee a maximum coverage.

Given an instance of $MaxCover(n, m, k)$ with $1 < k < m$, we can construct a network G with m nodes, each corresponds to one subset in $\mathcal{S}$. For each element e_i in $MaxCover(n, m, k)$, we construct a valid subgraph G_i as follows. First, we scan set $\mathcal{S}$ and obtain all the sets $\{\mathcal{B}_1^i, \ldots, \mathcal{B}_l^i\} \subseteq \mathcal{S}$ that contain element e_i. Then we map $\{\mathcal{B}_1^i, \ldots, \mathcal{B}_l^i\}$ into the nodes in G and get the corresponding l nodes. By connecting those l nodes with edges, we get a subgraph G_i in G with connectivity score $f(G_i) = w_i$. In this way, removing any nodes from G_i would break the

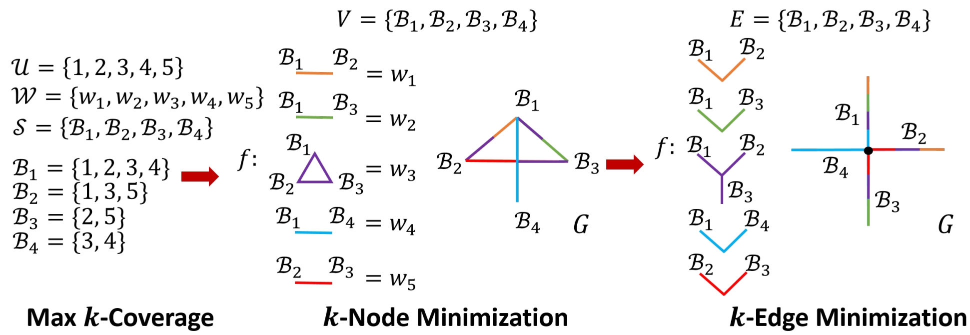


Figure 5.1: An Illustration of Polynomial Reduction from Max k-coverage Problem.

subgraph completeness. Repeating the above process for all the elements in $\mathcal{U}$, the final graph we get is $G = G_1 \cup \ldots \cup G_n$, and the connectivity function is defined as $f(G_i) = w_i$. Since the sets in $\mathcal{S}$ are distinct and not mutually exclusive, the resulting valid subgraphs are guaranteed to be non-independent. Therefore, the solution of $MaxCover(n, m, k)$ would be equivalent to the solution of $NodeMin(G, k)$, which proves that $NodeMin(G, k)$ is NP-hard.

On the other hand, given a candidate solution set of k node, we can verify its optimality by comparing to the $\binom{n}{k}$ all possible solution sets of size k. As calculating the impact of each set take polynomial time w.r.t. the network size and $\binom{n}{k}$ is also polynomial to the size of the network, the overall complexity to verify the optimality of the candidate solution is also polynomial to the size of the network, which indicates that $NodeMin(G, k)$ is in NP.

Therefore, the $NodeMin(G, k)$ problem is NP-complete. $\square$

Figure 5.1 gives an illustration of the reduction from an instance of max k-coverage problem $MaxCover(n, m, k)$ to k-node minimization problem $NodeMin(G, k)$, in which different valid subgraphs are marked with different colors. Edges with multiple colors indicate that they are involved in multiple valid subgraphs.

Lemma 8. *The k-edge connectivity minimization is NP-complete.*

Proof. We still use the connectivity measure defined Eq. (5.2) to complete the proof. The corresponding k-edge minimization problem can be defined as follows.

Problem 9. k-Edge Minimization Problem ($EdgeMin(G, k)$)

Given: *(1) A network $G =< V, E >$; (2) the connectivity function f as defined in Eq.* (5.2)*; and (3) the budget k.*

Output: *A set with k edges $E' \subseteq E$, such that $C(G \setminus E')$ (i.e., the connectivity in $G \setminus E'$) is minimized.*

We first prove that $EdgeMin(G, k)$ is NP-hard by constructing a polynomial reduction from $MaxCover(n, m, k)$ problem. Similar to the rationale in the previous proof, we assume that $1 < k < m$.

Given an instance of $MaxCover(n, m, k)$, we construct an m-edge star-shaped network G (i.e., all the m edges share one common endpoint). Specifically, each subset in $\mathcal{S}$ corresponds to an edge in G and each element e_i in $\mathcal{U}$ represents a valid subgraph G_i in the connectivity function. To construct subgraph G_i, we first locate all the subsets in $\mathcal{S}$ that contain element e_i, and map them into the corresponding edges in G. Then the sub-star formed by those edges can be viewed as a valid subgraph G_i with $f(G_i) = w_i$. The removal of any edge from G_i would destroy the completeness of the valid subgraph. Consequently, we have n valid subgraphs in G. Similarly, as the sets in $\mathcal{S}$ are distinct and not mutually exclusive, the resulting valid subgraphs are guaranteed to be non-independent. Therefore, the solution of $MaxCover(n, m, k)$ would be equivalent to the solution of $EdgeMin(G, k)$, which proves that $EdgeMin(G, k)$ is NP-hard.

Again, given a candidate solution set of k node, we can verify its optimality by comparing to the $\binom{m}{k}$ all possible solution sets of size k. As calculating the impact of each set take polynomial time w.r.t. the network size and $\binom{m}{k}$ is also polynomial to the size of the network, the overall complexity to verify the optimality of the candidate solution is also polynomial to the size of the network, which indicates that $EdgeMin(G, k)$ is in NP.

Therefore, the $EdgeMin(G, k)$ problem is NP-complete. □

Figure 5.1 gives an illustration of the reduction from an instance of max k-coverage problem $MaxCover(n, m, k)$ to k-edge minimization problem $EdgeMin(G, k)$. Again, edges with multiple colors indicate their participation in multiple valid subgraphs.

Fundamental Limit #2: Approximability

Based on the hardness result of NETCOM, we further reveal the approximability of NETCOM in Theorem 4, which says that $(1-1/e)$ is indeed the best approximation ratio a polynomial algorithm can achieve unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$).

Theorem 4. Approximability of NETCOM. $(1-1/e)$ *is the best approximation ratio for the* NETCOM *problem in polynomial time, unless NP* $\subseteq$ *DTIME*($n^{O(\log\log n)}$).

Proof. We prove this by contradiction. In the proof of Theorem 3, we show that max k-Coverage problem is polynomially reducible to the NETCOM problem, which implies that if there is an α-approximation algorithm that can solve NETCOM in polynomial time with $\alpha > (1-1/e)$, there will be an α-approximation algorithm for max k-Coverage as well. However, it has been proved in Khuller *et al.* (1999) that the maximum k-coverage problem can not be approximated with a factor better than $(1-1/e)$ unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$), which contradicts with our assumption. Hence, we conclude that there is no polynomial algorithm for the NETCOM problem with an approximation ratio greater than $(1-1/e)$, unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$).

□

Since the greedy strategy in Algorithm 8 guarantees a $(1 - 1/e)$ approximation ratio, Theorem 4 implies that the greedy algorithm is the best polynomial algorithm for NETCOM in terms of its approximation ratio unless NP $\subseteq$ DTIME$(n^{O(\log \log n)})$.

5.1.3 Proposed Algorithm

In this section, we start with detailing the algorithmic challenges of the network connectivity optimization (NETCOM) problem, and then present an effective algorithm, followed by some analysis in terms of its effectiveness and efficiency.

Algorithmic Challenges of NETCOM

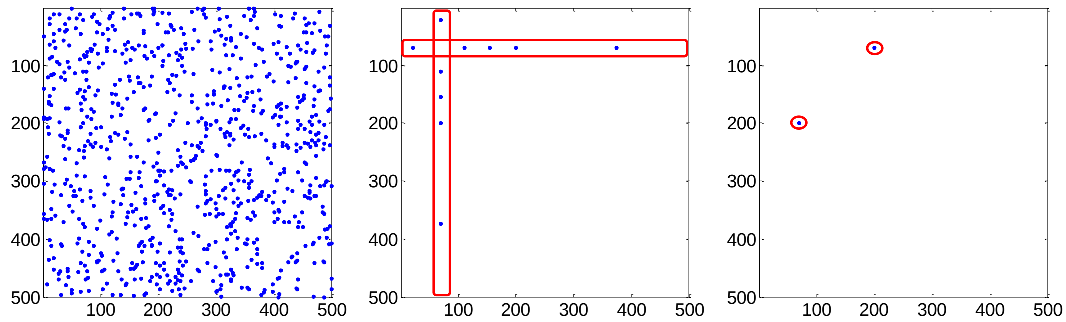
In the greedy strategy (Algorithm 8), a key step is to calculate the impact score of each network element, i.e., $I(o) = C(G) - C(G \setminus \{o\})$ (Step 4). As we have mentioned in Chapter 3, the network connectivity measures $C(G)$ studied in this thesis can be calculated or well approximated by a function of top-r eigenvalues of its adjacency matrix (i.e., $C(G) = F(\mathbf{\Lambda}^{(r)})$, where $F()$ is the function of eigenvalues). Therefore, the core step of calculating $I(o)$ is to compute $\mathbf{\Lambda}^{(r)}$ on $G \setminus \{o\}$, which takes $O(m)$ time (say using the classic Lanczos method). Consequently, simply recomputing $C(G \setminus \{o\})$ for each network element from scratch would make the entire algorithm $O(mn)$ for node-level optimization problems and $O(m^2)$ for edge-level optimization problems, neither of which is computationally feasible in large networks. To address this issue, existing literature often resorts to matrix perturbation theory. Its key idea is to view the deletion of a network element o as a perturbation to the original network (i.e., $\tilde{\mathbf{A}} = \mathbf{A} + \Delta\mathbf{A}$). Thus, the new eigenvalues (and hence the new connectivity measure $C(G \setminus \{o\})$) can be approximated from the eigenvalues and eigenvectors of the original network in constant time, making the overall algorithm linear w.r.t. the size of the input network Chen *et al.* (2016b,a). However, for networks with small eigen-gaps, the approximation accuracy of matrix perturbation theory based methods might deteriorate quickly, if not collapse at all. This issue might persist even if we switch to computationally more expensive high-order matrix perturbation theory Chen *et al.* (2016b,a). Thus, the main algorithmic challenge is how to accurately approximate the top-r eigenvalues of the input network after a node/edge deletion.

CONTAIN: The Proposed Algorithm

We propose a new updating algorithm for the top-r eigenvalues after node/edge deletion. In order to maintain the linear complexity of the entire greedy algorithm, we seek to update the top-r eigenvalues in *constant* time for each node/edge deletion operation.

Our key observation is as follows. In classic matrix perturbation theory (whether the first-order matrix perturbation theory or its high-order variants), a fundamental assumption is that the perturbation matrix $\Delta\mathbf{A}$ is a random matrix whose spectrum is well-bounded as illustrated in Figure 5.2(a). However, such assumption does not hold in the node/edge deletion scenario (Figure 5.2(b) and (c)), in which the perturbation matrix $\Delta\mathbf{A}$ is *sparse* and *low-rank*. Armed with this observation, we propose an effective eigen-pair update algorithm for node/edge deletion based on partial-QR decomposition. Unlike matrix perturbation based methods, which would inevitably introduce approximation error in the procedure, the proposed algorithm does not introduce any additional error when computing the impact score $I(o)$, and it runs in constant time for each node/edge operation.

The proposed CONTAIN algorithm is presented in Algorithm 9. Overall, it follows the greedy strategy (Algorithm 8). In detail, We first compute the top-r eigen-pairs of the network and compute the connectivity score of the original network (step 2-3). From step 4 to step 19, we iteratively select the element with the highest



(a) Random (b) Node Deletion (c) Edge Deletion

Figure 5.2: Illustrations and Comparison of Random Perturbation Matrix (a), Which Is Dense and Potentially Full-rank, Vs. Perturbation Matrices by Node Deletion (b) and Edge Deletion (c), Both of Which Are Sparse and Low-rank.

impact score. When evaluating the impact of each valid element, we first construct the perturbation matrix $\Delta\mathbf{A}$ for the corresponding element and then perform eigen-decomposition on it (step 6-7). Particularly, for node deletion operation, suppose the removed node v has a set of neighbor nodes $\mathcal{N}_v$. Then the resulting perturbation matrix $\Delta\mathbf{A}$ has $\Delta\mathbf{A}(v, \mathcal{N}_v) = \Delta\mathbf{A}(\mathcal{N}_v, v) = -1$, which is a rank-2 sparse matrix. Therefore, $\mathbf{U}_\Delta$ and $\mathbf{\Lambda}_\Delta$ can be directly expressed as an $n \times 2$ matrix and a 2×2 matrix respectively. Moreover, let $n_v = |\mathcal{N}_v|$, the non-zero entries in the eigenvector matrix of $\Delta\mathbf{A}$ are

$$\begin{aligned} \mathbf{U}_\Delta(v,1) &= \frac{1}{\sqrt{2}}, \quad \mathbf{U}_\Delta(v,2) = \frac{1}{\sqrt{2}} \\ \mathbf{U}_\Delta(\mathcal{N}_v,1) &= -\frac{1}{\sqrt{2n_v}}, \quad \mathbf{U}_\Delta(\mathcal{N}_v,2) = \frac{1}{\sqrt{2n_v}} \end{aligned} \tag{5.3}$$

and the eigenvalue matrix of $\Delta\mathbf{A}$ is

$$\mathbf{\Lambda}_\Delta = \begin{bmatrix} \sqrt{n_v} & 0 \\ 0 & -\sqrt{n_v} \end{bmatrix} \tag{5.4}$$

In the edge deletion scenario, the perturbation matrix $\Delta\mathbf{A}$ corresponding to the removal of edge $\langle u, v\rangle$ has only two non-zero entries $\Delta\mathbf{A}(u,v) = \Delta\mathbf{A}(v,u) = -1$ and

Algorithm 9 The CONTAIN Algorithm

Input: (1) The adjacency matrix of the network $\mathbf{A}$; (2) the associated eigen-function $F()$ for connectivity $C(G)$; (3) rank r; (4) the network operation (node vs. edge deletion); and (5) a positive integer k.

Output: a set of network elements $\mathcal{X}$ of size k.

initialize $\mathcal{X}$ to be empty
compute $[\mathbf{U}^{(r)}, \mathbf{\Lambda}^{(r)}] \leftarrow$top-$r$ eigen-pairs of matrix $\mathbf{A}$
compute $C(G) \leftarrow F(\mathbf{\Lambda}^{(r)})$
for $i = 1$ to k **do**
  for each valid element o in G **do**
    $\Delta\mathbf{A} \leftarrow$ the perturbation matrix by element o's deletion
    $[\mathbf{U}_\Delta, \mathbf{\Lambda}_\Delta] \leftarrow$eigen-pairs of $\Delta\mathbf{A}$
    $\mathbf{R} \leftarrow$upper triangular matrix from $[\mathbf{U}^{(r)}, \mathbf{U}_\Delta]$'s partial-QR decomposition
    $\mathbf{\Lambda}_z \leftarrow$ eigenvalues of $\mathbf{Z} = \mathbf{R}[\mathbf{\Lambda}^{(r)}, \mathbf{0}; \mathbf{0}, \mathbf{\Lambda}_\Delta]\mathbf{R}'$
    compute $I(o) \leftarrow C(G) - F(\mathbf{\Lambda}_z)$
  end for
  add $\tilde{o} = \text{argmax}_o I(o)$ to $\mathcal{X}$
  update $C(G) \leftarrow C(G) - I(\tilde{o})$ and set $I(\tilde{o}) \leftarrow -1$
  $\Delta\mathbf{A} \leftarrow$ the perturbation matrix by element $\tilde{o}$'s deletion
  $[\mathbf{U}_\Delta, \mathbf{\Lambda}_\Delta] \leftarrow$ eigen-pairs of $\Delta\mathbf{A}$
  $[\mathbf{Q}, \mathbf{R}] \leftarrow$ partial-QR decomposition of $[\mathbf{U}^{(r)}, \mathbf{U}_\Delta]$
  $[\mathbf{U}_z, \mathbf{\Lambda}_z] \leftarrow$ eigen-pairs of $\mathbf{Z} = \mathbf{R}[\mathbf{\Lambda}^{(r)}, \mathbf{0}; \mathbf{0}, \mathbf{\Lambda}_\Delta]\mathbf{R}'$
  update $\mathbf{U}^{(r)} \leftarrow (\mathbf{Q}\mathbf{U}_z)^{(r)}$, $\mathbf{\Lambda}^{(r)} \leftarrow \mathbf{\Lambda}_z^{(r)}$, $\mathbf{A} \leftarrow \mathbf{A} + \Delta\mathbf{A}$
end for
return $\mathcal{X}$

$u \neq v$, which is also a rank-2 matrix. Then, the only non-zero entries in $\mathbf{U}_\Delta$ are

$$\begin{aligned} \mathbf{U}_\Delta(u,1) &= \frac{1}{\sqrt{2}}, \quad \mathbf{U}_\Delta(u,2) = \frac{1}{\sqrt{2}} \\ \mathbf{U}_\Delta(v,1) &= -\frac{1}{\sqrt{2}}, \quad \mathbf{U}_\Delta(v,2) = \frac{1}{\sqrt{2}} \end{aligned} \tag{5.5}$$

And the eigenvalue matrix $\mathbf{\Lambda}_\Delta$ is

$$\mathbf{\Lambda}_\Delta = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{5.6}$$

With the eigenvector matrix of $\Delta\mathbf{A}$, we proceed to perform partial-QR decomposition on $[\mathbf{U}^{(r)}, \mathbf{U}_\Delta]$ in step 8. As $\mathbf{U}^{(r)}$ is already orthonormal, the $\mathbf{Q}$ matrix in the decomposition can be written as the concatenation of $\mathbf{U}^{(r)}$ and two orthogonal vectors in unit length as follows

$$\mathbf{Q} = [\mathbf{U}^{(r)}, \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|}, \frac{\mathbf{q}_2}{\|\mathbf{q}_2\|}] \tag{5.7}$$

By the Gram-Schmidt process, we have

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{U}_\Delta(:,1) - \mathbf{U}^{(r)}\mathbf{r}_1 \\ \mathbf{q}_2 &= \mathbf{U}_\Delta(:,2) - \mathbf{U}^{(r)}\mathbf{r}_2 + \mathbf{r}_1'\mathbf{r}_2\frac{\mathbf{q}_1}{\|\mathbf{q}_1\|^2} \end{aligned} \tag{5.8}$$

where $\mathbf{r}_1 = \mathbf{U}^{(r)\prime}\mathbf{U}_\Delta(:,1)$ and $\mathbf{r}_2 = \mathbf{U}^{(r)\prime}\mathbf{U}_\Delta(:,2)$.

For node-level operations, we have

$$\begin{aligned} \mathbf{r}_1 &= \mathbf{U}^{(r)\prime}\mathbf{U}_\Delta(:,1) = \frac{1}{\sqrt{2}}(\mathbf{U}^{(r)}(v,:) - \frac{1}{\sqrt{n_v}}\sum_{u\in\mathcal{N}_v}\mathbf{U}^{(r)}(u,:))' \\ \mathbf{r}_2 &= \mathbf{U}^{(r)\prime}\mathbf{U}_\Delta(:,2) = \frac{1}{\sqrt{2}}(\mathbf{U}^{(r)}(v,:) + \frac{1}{\sqrt{n_v}}\sum_{u\in\mathcal{N}_v}\mathbf{U}^{(r)}(u,:))' \end{aligned} \tag{5.9}$$

While for edge-level operations, we have

$$\begin{aligned} \mathbf{r}_1 &= \mathbf{U}^{(r)\prime}\mathbf{U}_\Delta(:,1) = \frac{1}{\sqrt{2}}(\mathbf{U}^{(r)}(u,:) - \mathbf{U}^{(r)}(v,:))' \\ \mathbf{r}_2 &= \mathbf{U}^{(r)\prime}\mathbf{U}_\Delta(:,2) = \frac{1}{\sqrt{2}}(\mathbf{U}^{(r)}(u,:) + \mathbf{U}^{(r)}(v,:))' \end{aligned} \tag{5.10}$$

Correspondingly, the upper-triangular matrix $\mathbf{R}$ can be written as

$$\mathbf{R} = \begin{bmatrix} \mathbf{I} & \mathbf{r}_1 & \mathbf{r}_2 \\ 0 & \|\mathbf{q}_1\| & -\frac{\mathbf{r}_1'\mathbf{r}_2}{\|\mathbf{q}_1\|} \\ 0 & 0 & \|\mathbf{q}_2\| \end{bmatrix} \tag{5.11}$$

By the definition of $\mathbf{q}_1, \mathbf{q}_2$ in Eq. (5.8) together with the orthonormal property of the eigenvectors, the norms of $\mathbf{q}_1$ and $\mathbf{q}_2$ can be computed indirectly with two $r \times 1$ vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ as

$$\begin{aligned} \|\mathbf{q}_1\| &= \sqrt{1 - \|\mathbf{r}_1\|^2} \\ \|\mathbf{q}_2\| &= \sqrt{1 - \|\mathbf{r}_2\|^2 - \frac{(\mathbf{r}_1'\mathbf{r}_2)^2}{1 - \|\mathbf{r}_1\|^2}} \end{aligned} \tag{5.12}$$

This enables us to compute $\|\mathbf{q}_1\|$ and $\|\mathbf{q}_2\|$ without explicitly constructing $\mathbf{q}_1$ and $\mathbf{q}_2$, which reduces the cost of step 8 from $O(nr)$ to $O(r)$. It can be proved that by setting $\mathbf{Z} = \mathbf{R}[\mathbf{\Lambda}^{(r)}, \mathbf{0}; \mathbf{0}, \mathbf{\Lambda}_\Delta]\mathbf{R}'$, the eigenvalues of $\mathbf{Z}$ are just the top eigenvalues of the perturbed matrix $\mathbf{A} + \Delta\mathbf{A}$, and the top eigenvectors of $\mathbf{A} + \Delta\mathbf{A}$ can be calculated by $\mathbf{QU}_z$ (step 18). Therefore, we only need $\mathbf{\Lambda}_z$ to compute the impact score of element o (step 10). After scanning all the valid elements in the current network, we choose the one with the largest impact score and add it to the element set $\mathcal{X}$ (step 12-13). Then, we update the network and its eigen-pairs (step 14-18). The procedure to update eign-pairs is similar to that of computing the impact score for a given network element (step 6-9), with the following subtle difference. In order to just compute the impact score of a given network element, we only need the updated eigenvalues. This is crucial as it saves the computation of (1) constructing $\mathbf{q}_1$ and $\mathbf{q}_2$, (2) finding the eigenvectors of $\mathbf{Z}$, and (3) updating the eigenvectors of the perturbed matrix $\mathbf{A} + \mathbf{\Delta A}$, which in turn helps maintain constant time complexity for each inner for-loop (step 5-11).

Algorithm 9 can be easily extended to address the local connectivity minimization problem by properly approximate the local connectivity impact score at step 10. It is worth to note that for some complex local connectivity measures like local natural connectivity or local length-t path capacity, it is often time-consuming to directly calculate local connectivity $C_{\mathcal{T}}(G)$ and element impact $I_{\mathcal{T}}(o)$ on $C_{\mathcal{T}}(G)$ than the global connectivity $C(G)$ and $I(o)$. This is mainly because that $C_{\mathcal{T}}(G)$ and $I_{\mathcal{T}}(o)$ can not be directly calculated with the eigen-pairs of the network. To efficiently address this problem, we propose the following heuristic to measure $C_{\mathcal{T}}(G)$ and $I_{\mathcal{T}}(o)$.

Lemma 9. Local Impact Computation. *Let $C(G \setminus \mathcal{T})$ and $I^{G\setminus\mathcal{T}}(o)$ denotes the global connectivity of graph $G \setminus \mathcal{T}$ and impact of element o on $C(G \setminus \mathcal{T})$, then $C_{\mathcal{T}}(G) = C(G) - C(G \setminus \mathcal{T})$, and $I_{\mathcal{T}}(o) = I(o) - I^{G\setminus\mathcal{T}}(o)$.*

Proof. The first equation naturally holds by the definition of connectivity measures. Here we proceed to prove the second part. By the definition of $I_{\mathcal{T}}(o)$, we have

$$I_{\mathcal{T}}(o) = C_{\mathcal{T}}(G) - C_{\mathcal{T}}(G \setminus \{o\})$$

By the fact that $C_{\mathcal{T}}(G) = C(G) - C(G \setminus \mathcal{T})$, the above equation can be re-write as

$$\begin{aligned} I_{\mathcal{T}}(o) &= (C(G) - C(G \setminus \mathcal{T})) - (C(G \setminus \{o\}) - C(G \setminus \{o\} \cup \mathcal{T}) \qquad (5.13) \\ &= (C(G) - C(G \setminus \{o\})) - (C(G \setminus \mathcal{T}) - C(G \setminus \{o\} \cup \mathcal{T}) \\ &= I(o) - I^{G\setminus\mathcal{T}}(o) \end{aligned}$$

□

Proof and Analysis

In this section, we analyze the proposed CONTAIN algorithm w.r.t. its effectiveness and efficiency.

Effectiveness. The effectiveness of CONTAIN is summarized in Lemma 10, which says that the computation of the impact score for each valid network element in the inner for-loop does not introduce any extra approximation error.

Lemma 10. Effectiveness of CONTAIN. *Suppose* $\mathbf{A}$ *is approximated with its top-r eigen-pairs with error* $\mathbf{E}$ *(i.e.,* $\mathbf{A} = \mathbf{U}^{(r)}\mathbf{\Lambda}^{(r)}\mathbf{U}^{(r)'} + \mathbf{E}$*), then the* $\mathbf{\Lambda}_z$ *and* $\mathbf{QU}_z$ *returned in Algorithm 9 can be used to approximate* $\tilde{\mathbf{A}}$ *as its top eigen-pairs with no extra error.*

Proof. As $\mathbf{A} = \mathbf{U}^{(r)}\mathbf{\Lambda}^{(r)}\mathbf{U}^{(r)'} + \mathbf{E}$ and $\Delta\mathbf{A} = \mathbf{U}_\Delta\mathbf{\Lambda}_\Delta\mathbf{U}'_\Delta$, then $\tilde{\mathbf{A}}$ can be expressed as

$$\begin{aligned}\tilde{\mathbf{A}} &= \mathbf{U}^{(r)}\mathbf{\Lambda}^{(r)}\mathbf{U}^{(r)'} + \mathbf{U}_\Delta\mathbf{\Lambda}_\Delta\mathbf{U}'_\Delta + \mathbf{E} \\ &= [\mathbf{U}^{(r)}, \mathbf{U}_\Delta] \begin{bmatrix} \mathbf{\Lambda}^{(r)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}_\Delta \end{bmatrix} [\mathbf{U}^{(r)}, \mathbf{U}_\Delta]' + \mathbf{E} \end{aligned} \tag{5.14}$$

Perform partial-QR decomposition on $[\mathbf{U}^{(r)}, \mathbf{U}_\Delta]$ as $[\mathbf{U}^{(r)}, \mathbf{U}_\Delta] = \mathbf{QR}$, we get orthonormal basis for $\tilde{\mathbf{A}}$ and an upper triangular matrix $\mathbf{R}$. Then the perturbed matrix $\tilde{\mathbf{A}}$ can be rewritten as

$$\tilde{\mathbf{A}} = \mathbf{QR} \begin{bmatrix} \mathbf{\Lambda}^{(r)} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}_\Delta \end{bmatrix} \mathbf{R}'\mathbf{Q}' + \mathbf{E} \tag{5.15}$$

Let $\mathbf{Z} = \mathbf{R}[\mathbf{\Lambda}^{(r)}, \mathbf{0}; \mathbf{0}, \mathbf{\Lambda}_\Delta]\mathbf{R}'$ and perform eigen-decomposition on $\mathbf{Z}$ as $\mathbf{Z} = \mathbf{U}_z\mathbf{\Lambda}_z\mathbf{U}'_z$, $\tilde{\mathbf{A}}$ is now equivalent to

$$\tilde{\mathbf{A}} = \mathbf{QU}_z\mathbf{\Lambda}_z\mathbf{U}'_z\mathbf{Q}' + \mathbf{E} = (\mathbf{QU}_z)\mathbf{\Lambda}_z(\mathbf{QU}_z)' + \mathbf{E} \tag{5.16}$$

Since both $\mathbf{Q}$ and $\mathbf{U}_z$ are orthonormal, we have $(\mathbf{QU}_z)(\mathbf{QU}_z)' = \mathbf{I}$. Thus, $\mathbf{\Lambda}_z$ and $\mathbf{QU}_z$ can be viewed as the top eigen-pairs of $\tilde{\mathbf{A}}$. As the approximation error remains to be $\mathbf{E}$ in Eq. (5.16), it implies that no extra error is introduced in the procedure, which completes the proof. □

As the eigenvalues in real networks are often skewed Faloutsos *et al.* (1999), the above impact scores can be approximated with top-r eigenvalues. Analysis in existing literature (Tsourakakis (2008) and Chan *et al.* (2014)) show that the truncated approximations for triangle capacity and natural connectivity can achieve high accuracy with only top-50 eigenvalues, which enables a great acceleration on impact score approximation.

Efficiency. The complexity of the proposed CONTAIN algorithm is summarized in Lemma 11, which says it is linear in both time and space.

Lemma 11. Complexity of CONTAIN. *The time complexity of* CONTAIN *for node-level connectivity optimization is* $O(k(mr + nr^3))$. *The time complexity of* CONTAIN *for edge-level connectivity optimization is* $O(k(mr^3 + nr^2))$. *The space complexity of* CONTAIN *is* $O(nr + m)$.

Proof. In the CONTAIN algorithm, computing top-r eigen-pairs and connectivity $C(G)$ would take $O(nr^2+mr)$ and $O(r)$ respectively. To compute the impact score for each node/edge (step 5-11), it takes $O(d_v r)$ (d_v is the degree of node v) for node v, and $O(r)$ for each edge to get the upper triangular matrix $\mathbf{R}$ in step 8. Since performing eigen-decomposition on $\mathbf{Z}$ at step 9 takes $O(r^3)$, the complexity to collect impact scores for all the nodes/edges are $O(nr^3 + mr)$ and $O(mr^3)$ respectively. Picking out the node/edge with highest impact score in current iteration would cost $O(n)$ for node level operations and $O(m)$ for edge level operations. At the end of the iteration, updating the eigen-pairs of the network takes the complexity of $O(nr^2 + r^3)$. As we have $r \ll n$, the overall time complexity to select k nodes would be $O(k(mr + nr^3))$; and the complexity to select k edges would be $O(k(mr^3 + nr^2))$

For space complexity, it takes $O(n + m)$ to store the entire network, $O(nr)$ to calculate and store the top-r eigen-pair of the network, $O(n)$ to store the impact scores for all the nodes in node level optimization scenarios and $O(m)$ to store the impact scores for all the edges, the eigen-pair update requires a space of $O(nr)$. Therefore, the overall space complexity for CONTAIN is $O(nr+m)$. $\square$

CONTAIN+: The Closed-form Heuristics

Here we provide the heuristics for the triangle capacity and natural connectivity optimization problems, which can be easily extended to other similar connectivity measures.

Impact Approximation. For triangle capacity optimization problem, the impact of a node/edge is the number of triangles that the node/edge participates in, which can be directly approximated with the eigen-pairs of the current network.

Lemma 12. Closed-Form Impact Score for Triangle Capacity. *Given a network G with adjacency matrix $\mathbf{A}$ and eigen-pair $(\mathbf{U}, \mathbf{\Lambda})$. The number of triangles that node v participates in is $I(v) = \sum_{i=1}^{n} \frac{\lambda_i^3 \mathbf{u}_i^2(v)}{2}$; the number of triangles that edge $\langle u, v\rangle$ participates in is $I(\langle u, v\rangle) = \sum_{i=1}^{n} \lambda_i^2 \mathbf{u}_i(u)\mathbf{u}_i(v)$.*

Proof. The first part of the lemma has been proved in Tsourakakis (2008). Here we proceed to prove the second part.

The number of triangles that edge $\langle u, v\rangle$ involves in equals to the number of length-2 paths from node u to node v, which equals $\mathbf{A}^2(u, v)$. As $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}'$, we have $\mathbf{A}^2 = \mathbf{U}\Lambda^2\mathbf{U}'$. Therefore, $\mathbf{A}^2(u, v) = \sum_{i=1}^{n} \lambda_i^2 \mathbf{u}_i(u)\mathbf{u}_i(v)$. $\square$

Lemma 13. Closed-Form Impact Score for Natural Connectivity. *Given a network G with adjacency matrix $\mathbf{A}$ and eigen-pair $(\mathbf{U}, \mathbf{\Lambda})$. The impact of node v on natural connectivity is $I(v) = \sum_{i=1}^{n} e^{\lambda_i}\mathbf{u}_i^2(v)$; the impact of edge $\langle u, v\rangle$ is $I(\langle u, v\rangle) = \sum_{i=1}^{n} \frac{e^{\lambda_i}-1}{\lambda_i}\mathbf{u}_i(u)\mathbf{u}_i(v)$.*

Proof. Natural connectivity can be viewed as an aggregation of weighted closed-walks Jun *et al.* (2010). For node v, its impact on the number of length-t closed-walks is proportional to $\mathbf{A}^t(v,v)$, so its overall impact on natural connectivity can be expressed as $I(v) = \sum_{j=0}^{\infty} \frac{\mathbf{A}^j(v,v)}{j!}$. As we have $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}'$ and $\mathbf{A}^j(v,v) = \sum_{i=1}^{n} \lambda_i^j \mathbf{u}_i^2(v)$, we have

$$\begin{aligned} I(v) &= \sum_{j=0}^{\infty} \frac{1}{j!} \sum_{i=1}^{n} \lambda_i^j \mathbf{u}_i^2(v) = \sum_{i=1}^{n} \sum_{j=1}^{\infty} \frac{1}{j!} \lambda_i^j \mathbf{u}_i^2(v) \\ &= \sum_{i=1}^{n} e^{\lambda_i} \mathbf{u}_i^2(v) \end{aligned} \tag{5.17}$$

For edge $\langle u, v \rangle$, the number of length-t closed-walks it participates in equals to the number of length-$(t-1)$ walks from node u to node v, which can be expressed as $\mathbf{A}^{t-1}(u,v)$. Therefore, its overall impact on natural connectivity can be written as $I(\langle u,v \rangle) = \sum_{j=1}^{\infty} \frac{\mathbf{A}^{j-1}(u,v)}{j!}$. Let $\mathbf{T} = \sum_{j=1}^{\infty} \frac{\mathbf{A}^{j-1}}{j!}$, then we have

$$\mathbf{A}\mathbf{T} = \sum_{j=1}^{\infty} \frac{\mathbf{A}^j}{j!} = \sum_{j=0}^{\infty} \frac{\mathbf{A}^j}{j!} - \mathbf{I} = e^{\mathbf{A}} - \mathbf{I}$$

Based on the above equation, we have

$$\begin{aligned} \mathbf{T} &= \mathbf{A}^{-1}(e^{\mathbf{A}} - \mathbf{I}) = \mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}'(\mathbf{U}(e^{\mathbf{\Lambda}} - \mathbf{I})\mathbf{U}') \\ &= \mathbf{U}\mathbf{\Lambda}^{-1}(e^{\mathbf{\Lambda}} - \mathbf{I})\mathbf{U}' \end{aligned} \tag{5.18}$$

Thus, $I(\langle u,v \rangle) = \mathbf{T}(u,v) = \sum_{i=1}^{n} \frac{e^{\lambda_i} - 1}{\lambda_i} \mathbf{u}_i(u)\mathbf{u}_i(v)$ $\square$

Effectiveness of CONTAIN+. As we have mentioned, the impact score of a node/edge is often approximated with the top-r eigen-pairs of the network. Let $(\mathbf{U}, \mathbf{\Lambda})$ be the eigen-pairs of network G, $(\tilde{\mathbf{U}}, \tilde{\mathbf{\Lambda}})$ be the eigen-pairs of network $G \setminus \{v\}$. To estimate the impact of node v on the connectivity of the network, CONTAIN needs to utilize the eigenvalues from both the original network (i.e., $\mathbf{\Lambda}$) and the perturbed network (i.e., $\tilde{\mathbf{\Lambda}}$) for the calculation (Algorithm 9 step 10); while CONTAIN+ only relies on the eigen-pairs in the original network (i.e., $(\mathbf{U}, \mathbf{\Lambda})$). Take triangle capacity optimization under node operations as an example. the impact of node v can be approximated as $I(v)_{\text{CONTAIN}} = \sum_{i=1}^{r} \frac{\lambda_i^3}{6} - \sum_{i=1}^{r} \frac{\tilde{\lambda}_i^3}{6}$ by CONTAIN; or it can be approximated as $I(v)_{\text{CONTAIN+}} = \sum_{i=1}^{r} \frac{\lambda_i^3 \mathbf{u}_i^2(v)}{2}$ by CONTAIN+. Suppose the exact impact of node v is $I(v)_{Exact}$, then we can define the approximation error of CONTAIN as $err_{\text{CONTAIN}} = I(v)_{Exact} - I(v)_{\text{CONTAIN}}$ and the error of CONTAIN+ as $err_{\text{CONTAIN+}} = I(v)_{Exact} - I(v)_{\text{CONTAIN+}}$. In Lemma 14, we give the analysis on err_{CONTAIN} and $err_{\text{CONTAIN+}}$.

Lemma 14. *The error of* CONTAIN *for triangle capacity impact approximation is* $err_{\text{CONTAIN}} = \frac{\sum_{i=r+1}^{n} \lambda_i^3 - \tilde{\lambda}_i^3}{6}$*; and the approximation error for* CONTAIN+ *is* $err_{\text{CONTAIN+}} = \sum_{i=r+1}^{n} \frac{\lambda_i^3 \mathbf{u}_i^2(v)}{2}$.

Proof. As we have $I(v)_{Exact} = \sum_{i=1}^{n} \frac{\lambda_i^3 \mathbf{u}_i^2(v)}{2} = \frac{\sum_{i=1}^{n} \lambda_i^3 - \tilde{\lambda}_i^3}{6}$, Lemma 14 naturally holds when $I(v)_{\text{CONTAIN}}$ and $I(v)_{\text{CONTAIN+}}$ are subtracted from $I(v)_{Exact}$ respectively. □

Lemma 14 implies that when the removed node has small effect to the bottom-$(n-r)$ eigenvalues of the underlying network, CONTAIN is preferred as err_{CONTAIN} would be small. While in networks with very skewed eigenvalue distributions, CONTAIN+ is preferred as the bottom-$(n-r)$ eigenvalues are small in magnitude. Similar analysis can be derived for edge-level operation scenarios and natural connectivity optimization scenarios, which is omitted for brevity.

Complexity of CONTAIN+. In Theorem 5, we give the time and space complexity of CONTAIN+.

Theorem 5. *The time complexity of* CONTAIN+ *for node level connectivity optimization is* $O(nr^2 + mr + knr^2)$*. The time complexity for edge level connectivity optimization is* $O(k(mr+nr^2))$*. The space complexity of* CONTAIN+ *is* $O(nr+m)$*.*

Proof. The CONTAIN+ algorithm is generally similar to the CONTAIN algorithm except for the node/edge impact calculation part. Therefore, CONTAIN+ also need to take $O(nr^2 + mr)$ to compute the top-r eigen-pairs. In each iteration, it takes CONTAIN+ $O(nr)$ time to get all the impact scores for each node or $O(mr)$ for the scores of each edge. After picking out the highest impact node/edge with $O(n)$ or $O(m)$ complexity, we update the eigen-pairs of the network with the method used in CONTAIN with $O(nr^2 + r^3)$ complexity. Therefore, the overall complexity for node level connectivity minimization is $O(nr^2 + mr + knr^2)$ and the complexity for edge level connectivity minimization is $O(nr^2 + mr + k(mr + nr^2))$, which can be simplified as $O(k(mr + nr^2))$.

As for the space complexity, CONTAIN+ also need space to store all the top-r eigen-pairs and the impact scores for nodes/edges. Since no extra computation space is introduced in CONTAIN+ compared to CONTAIN, the overall space complexity for CONTAIN+ is still $O(nr + m)$. □

5.1.4 Experimental Evaluation

In this section, we evaluate the proposed CONTAIN algorithm. All experiments are designed to answer the following two questions:

- **Effectiveness.** How effective is the proposed CONTAIN algorithm in minimizing various connectivity measures?
- **Efficiency.** How efficient and scalable is the proposed CONTAIN algorithm?

Experiment Setup

Datasets. We perform experiments on 10 different datasets from 4 different domains, including AIRPORT: an air traffic network that represents the direct flight connections between internal US airports [4] ; OREGON: an autonomous system network which de-

[4] `http://www.levmuchnik.net/Content/Networks/NetworkData.html`.

picts the information transferring relationship between routers from Leskovec *et al.* (2005); CHEMICAL: a network based on Davis *et al.* (2015) that shows the similarity between different chemicals; DISEASE: a network that depicts the similarity between different diseases Davis *et al.* (2015); GENE: a protein-protein interaction network based on Davis *et al.* (2015); ASTRPH: a collaboration network between authors whose papers were submitted to Astro Physics category on Arxiv Leskovec *et al.* (2007b); HEPTH: a collaboration network between authors whose papers were submitted to High Energy Physics (Theory category) on Arxiv Leskovec *et al.* (2005); AMINER: a collaboration network between researchers in the Aminer datasets Tang *et al.* (2008); EUCORE: the email correspondence network from a large European research institution Leskovec *et al.* (2007b); and FB: a social circle network collected from Facebook Mcauley and Leskovec (2014). The statistics of those datasets are listed in Table 5.2.

Table 5.2: Statistics of Datasets.

Domain	**Dataset**	**#Nodes**	**#Edges**	**Avg Degree**
Infrastructure	AIRPORT	2,833	7,602	5.37
	OREGON	5,296	10,097	3.81
Biology	CHEMICAL	6,026	69,109	22.94
	DISEASE	4,256	30,551	14.36
	GENE	7,604	14,071	3.7
Collaboration	ASTRPH	18,772	198,050	21.1
	HEPTH	9,877	25,985	5.26
	AMINER	1,211,749	4,756,194	7.85
Social	EUCORE	1,005	16,064	31.97
	FB	4,039	88,234	43.69

Comparing Methods. We compare the proposed algorithm with the following methods. (1) *Degree*: selecting top–k nodes (edges) with the largest degrees; specifically, for edge $\langle u, v \rangle$, let d_u and d_v denote the degrees for its endpoints respectively, the score for $\langle u, v \rangle$ is $\min\{d_u, d_v\}$ [5] . (2) *PageRank*: selecting top–k nodes (edges) with the largest PageRank scores Page *et al.* (1998) (the corresponding edge score is the minimum PageRank score among its two endpoints); (3) *Eigenvector*: selecting top–k nodes (edges) with the largest eigenvector centrality scores Newman (2008) (the corresponding edge score is the minimum eigenvector centrality score among its endpoints); (4) *Netshield/Netmelt*: selecting top–k nodes (edges) that minimize the leading eigenvalue of the network Chen *et al.* (2016b,a); (5) *MIOBI*: a greedy algorithm that employs first-order matrix perturbation method to estimate element impact score and update eigen-pairs Chan *et al.* (2014); (6) *MIOBI-S*: a variant of MIOBI that selects top–k nodes (edges) in one batch without updating the eigen-pairs of the network; (7) *MIOBI-H*: a variant of MIOBI that employs high-order matrix perturbation method to update eigen-pairs Chen and Tong (2017); (8) *Exact*: a greedy algorithm that recomputes the top-r eigen-pairs to estimate the impact score for each candidate node/edge. For the results reported here, we set rank $r = 80$ for all the top–r eigen-pairs based approximation methods (methods (5)-(8) and the proposed CONTAIN method).

Evaluation Metrics. The performance of the algorithm is evaluated by the impact of its selected elements $I(\mathcal{X}) = C(G) - C(G \setminus \mathcal{X})$. The larger the $I(\mathcal{X})$ is, the more effective the algorithm is. For a given dataset, connectivity measure and network operation, we normalize $I(\mathcal{X})$ by that of the best method, so that the results across different datasets are comparable in the same plot.

[5] We use $\min\{d_u, d_v\}$ as edge score to ensure that both ends of the top-ranked edges are high degree nodes.

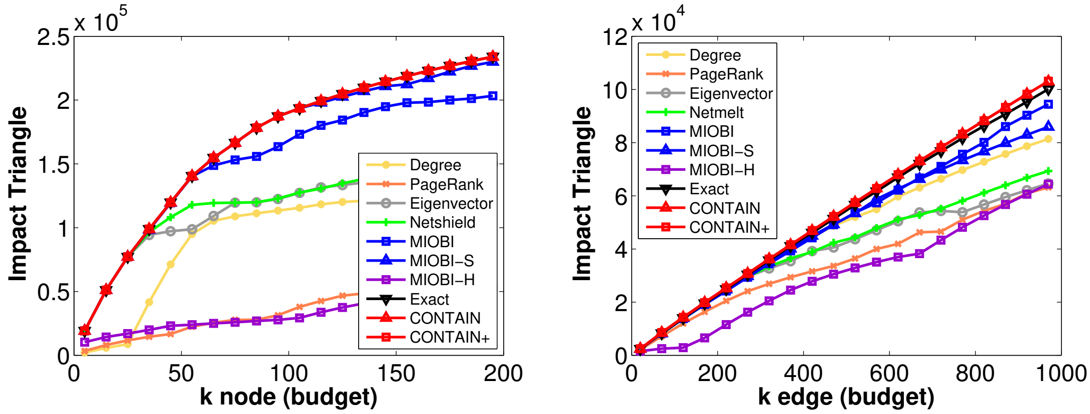
Machine and Repeatability All the experiments in this work are performed on a machine with 2 processors (Intel Xeon 3.5GHz) with 256GB of RAM. The algorithms are programmed with MATLAB using a single thread.

Effectiveness

Effectiveness of CONTAIN and CONTAIN+. We compare the proposed algorithm and the baseline methods on three connectivity measures (leading eigenvalue, number of triangles, and natural connectivity) by both node-level operations and edge-level operations on all datasets in our experiment. Since the *Exact* method needs to recompute the top-r eigen-pairs for each candidate node/edge which is very time-consuming, its results would be absent on some large datasets (e.g., AMINIER and ASTRPH) where it does not finish the computation within 24 hours. In our experiment, the budget for node-level operations is $k = 20$, the budget for edge-level operations is $k = 200$. The results are shown from Figure 5.4 to Figure 5.9. We can see that the proposed CONTAIN (the red solid bar) and CONTAIN+ (the red hollow bar) (1) are very close to the *Exact* method (the black hollow bar); and (2) consistently outperforms all the other alternative methods. In the meanwhile, the proposed CONTAIN and CONTAIN+ algorithms are much faster than *Exact*, as will be shown later.

To study the effectiveness of CONTAIN and CONTAIN+ for the local connectivity minimization problem. We experiment on the CHEMICAL data and compare the performance of different methods for minimizing the local triangle capacity in the network. From Figure 5.3, it is obvious to see that both CONTAIN and CONTAIN+ can achieve similar performance with the *Exact* algorithm and outperform all other heuristic methods.

Effect of Rank r. The main parameter that affects the performance of CONTAIN



(a) Node Operations. (b) Edge Operations.

Figure 5.3: The Optimization Results on the Number of Local Triangles on the CHEMICAL Dataset.

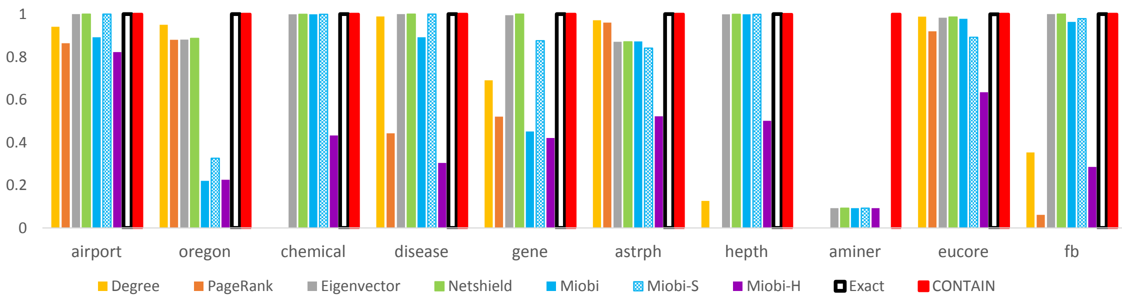


Figure 5.4: The Optimization Results on Leading Eigenvalue with Node-level Operations.

is the rank r. To study the effect of r, we change r from 5 to 80 to minimize the number of triangles on the CHEMICAL dataset and compare them with the *Exact* method. The results are shown in Figure 5.10. From Figure 5.10, it is obvious to see that as r increases, the performance of CONTAIN increases accordingly, which is consistent with our effectiveness analysis. With $r = 80$, the performance of CONTAIN is very close to the *Exact* method with different k.

Efficiency

Efficiency of CONTAIN. Figure 5.11 presents the quality vs. running time trade-off of different methods for optimizing the natural connectivity (the most complicated connectivity measure) on the EUCORE dataset. In both node-level and edge-level

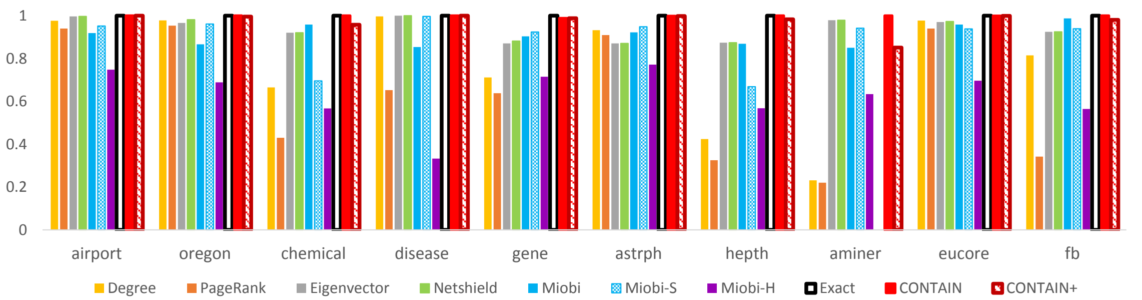


Figure 5.5: The Optimization Results on the Number of Triangles with Node-level Operations.

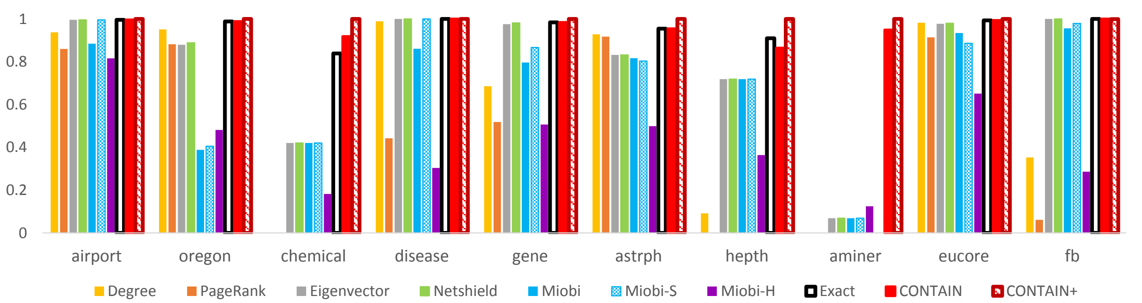


Figure 5.6: The Optimization Results on Natural Connectivity with Node-level Operations.

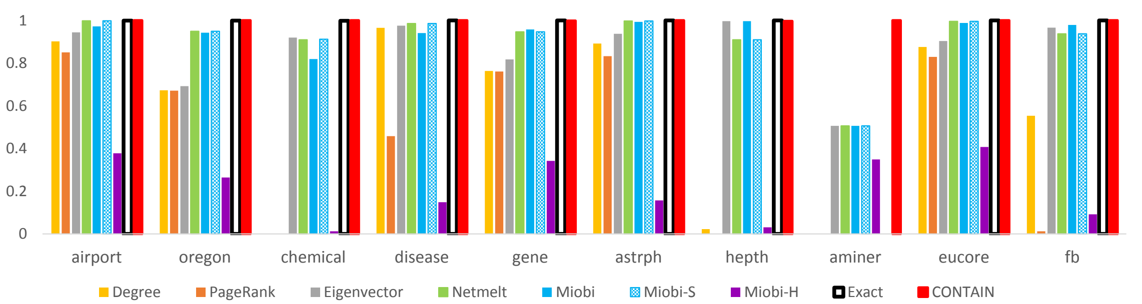


Figure 5.7: The Optimization Results on Leading Eigenvalue with Edge-level Operations.

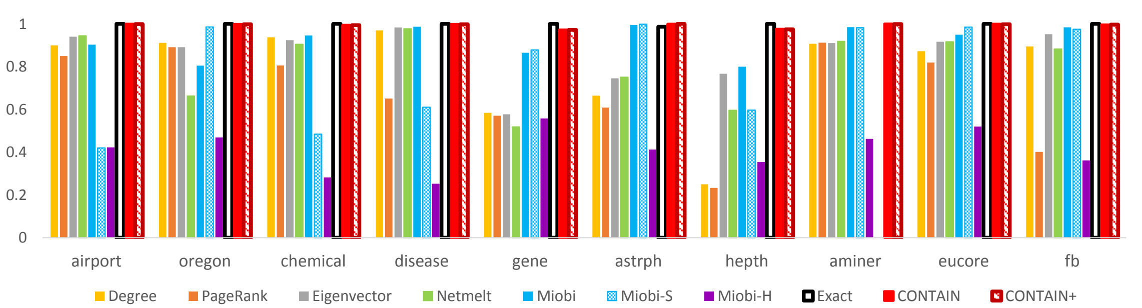


Figure 5.8: The Optimization Results on the Number of Triangles with Edge-level Operations.

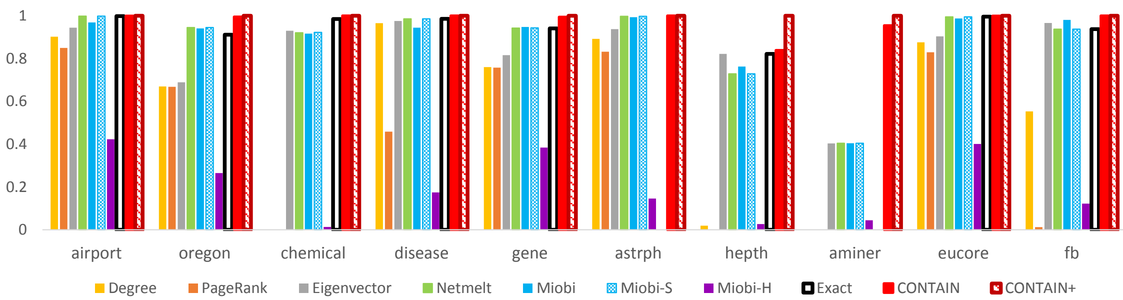
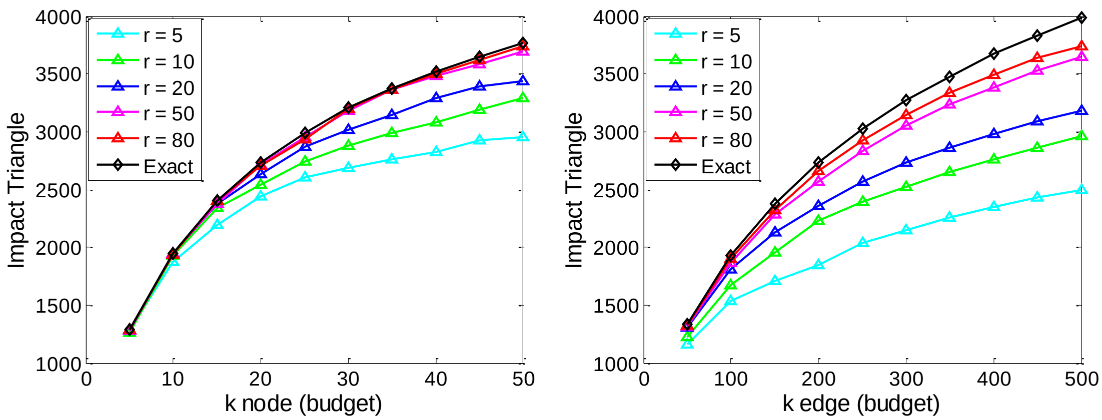


Figure 5.9: The Optimization Results on Natural Connectivity with Edge-level Operations.

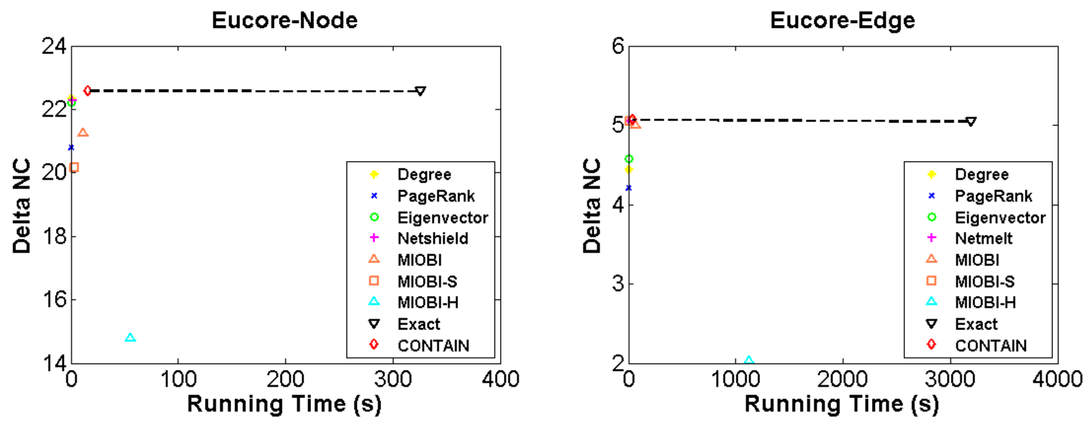


(a) Node Operations. (b) Edge Operations.

Figure 5.10: The Effect of r on Optimizing the Number of Triangles on CHEMICAL Dataset.

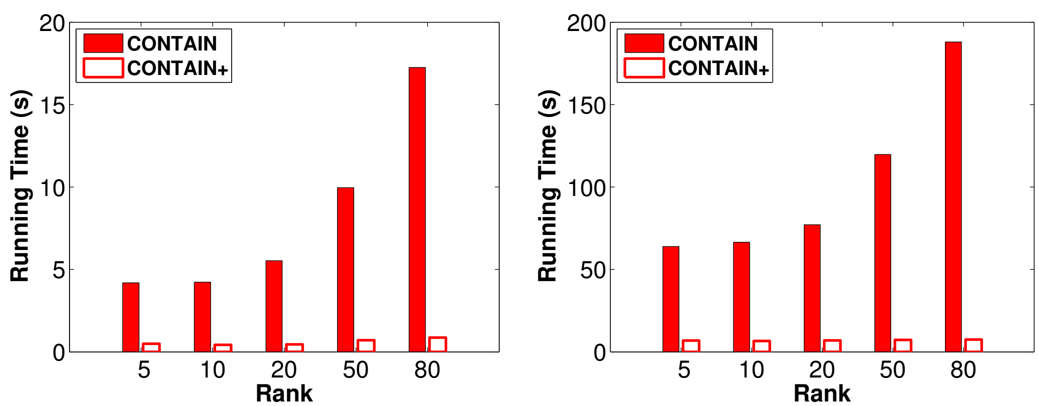
optimization scenarios, the proposed CONTAIN achieves very similar performance as *Exact*. In terms of the running time, CONTAIN is orders of magnitude faster than *Exact*. Although the running time of other baseline methods is similar to CONTAIN, their performance (y-axis) is not as good as CONTAIN.

Efficiency of CONTAIN+. To justify the efficiency of CONTAIN+, we compare the running time of CONTAIN and CONTAIN+ on the CHEMICAL dataset in Figure 5.12. We can see that the running time of CONTAIN+ is orders of magnitudes faster than the CONTAIN algorithm. Moreover, as rank r increases, the running time of CONTAIN would increase in polynomial order due to the eigen-



(a) Node Operations (b) Edge Operations

Figure 5.11: The Quality Vs. Running Time Trade-off on EUCORE. The Budget for Node Operations Is $k = 20$, the Budget for Edge Operations Is $k = 200$.

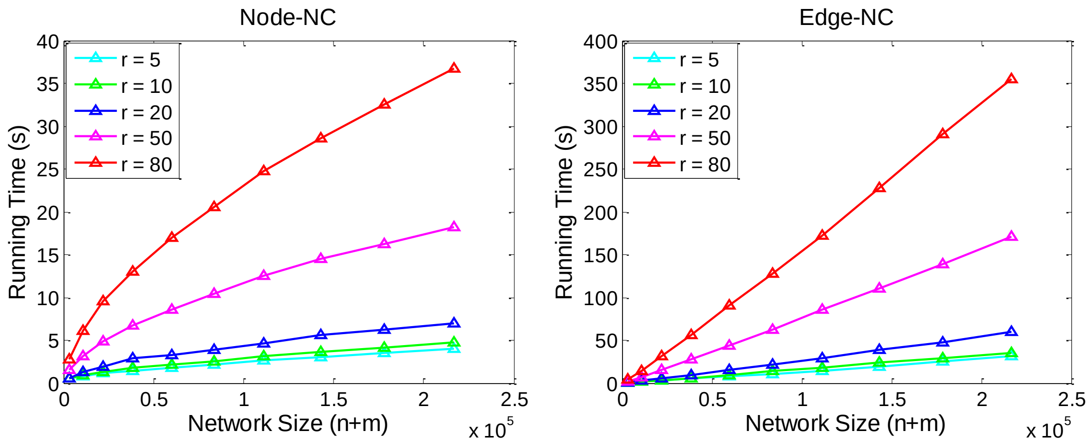


(a) Node Operations (b) Edge Operations

Figure 5.12: The Running Time Comparison Between CONTAIN and CONTAIN+ on the CHEMICAL Dataset. The Budget for Both Node and Edge Operations Is $k = 20$.

decomposition operation for node/edge impact score approximation; while the running time of CONTAIN+ only shows a slightly linear increase across different rank settings.

Scalability of CONTAIN. The scalability results of CONTAIN are presented in Figure 5.13. As we can see, the proposed CONTAIN algorithm scales linearly w.r.t. the size of the input network (i.e., both the number of nodes and edges), which is



(a) Node Operations (b) Edge Operations

Figure 5.13: The Scalability of CONTAIN. The Budget for Both Node and Edge Operations Is $k = 20$.

consistent with Lemma 11.

5.2 Connectivity Optimization in Multi-layered Networks

The multi-layered networks are fundamentally different from the single-layered networks due to the cross-layer dependencies between different networks. Such dependency has made multi-layered networks are more vulnerable to external attacks because their nodes can be affected by both within-layer connections and cross-layer dependencies. That is, even a small disturbance in one layer/network may be amplified in all its dependent networks through cross-layer dependencies, and cause cascade failure to the entire system. For example, when the supporting facilities (e.g., power stations) in a metropolitan area are destroyed by natural disasters like hurricanes or earthquakes, the resulting blackout would not only put tens of thousands of people in dark for a long time, but also paralyze the telecom network and cause a great interruption on the transportation network. Therefore, it is of key importance to identify crucial nodes in the supporting layer/network, whose loss would lead to a catastrophic failure of the entire system, so that the countermeasures can be taken proactively. On the other hand, accessibility issues extensively exist in multi-layered network mining tasks. To manipulate the connectivity in layers with limited accessibility, one can only operate via the nodes from accessible layers that have large impact to target layers. Taking the multi-layered network depicted in Figure 3.1(a) for example, assume that the only accessible layer in the system is the control layer and the goal is to minimize the connectivity in the satellite communication layer and physical layer simultaneously under k attacks, the only strategy we could adopt is to select a set of k nodes from the control layer, whose failure would cause largest reduction on the connectivity of the two target layers.

To tackle the connectivity optimization [6] problem in multi-layered networks, great efforts have been made from different research area for manipulating *two-layered* interdependent network systems Buldyrev *et al.* (2010); Parshani *et al.* (2010); Sen *et al.* (2014); Shao *et al.* (2011); Gao *et al.* (2012). Although much progress has been made, the optimization strategies used in two-layered networks may still be sub-optimal or even misleading in multi-layered network settings, where we want to simultaneously optimize the connectivity in multiple layers by manipulating one common supporting layer. On the theoretic side, the optimality of the connectivity optimization problem of generic multi-layered networks is largely unknown.

This work aims to address the above challenges, and the main contributions can be summarized as

- *Connectivity Optimization*. We show that for *any* network connectivity measures in the SubLine family, the connectivity optimization problem with the proposed MuLaN model enjoys the diminishing returns property, which naturally lends itself to a family of provable near-optimal optimization algorithms

[6] In this work, connectivity optimization problem is defined as minimizing the connectivity of a target layer by removing a fixed number of nodes in the control layer.

with linear complexity.

- *Empirical Evaluations.* We perform extensive experiments based on real data sets to validate the effectiveness and efficiency of the proposed algorithms.

5.2.1 Problem Definition

In this section, we start with the main symbols used in this work (Table 5.3) and then give the formal definition of multi-layered network connectivity optimization problem. We use bold upper case letters for matrices (e.g., $\mathbf{A}$, $\mathbf{B}$), bold lower case letters for column vectors (e.g., $\mathbf{a}$, $\mathbf{b}$) and calligraphic font for sets (e.g., $\mathcal{A}$, $\mathcal{B}$). The transpose of a matrix is denoted with a prime, i.e., $\mathbf{A}'$ is the transpose of matrix $\mathbf{A}$.

With the above symbols [7] and the definition of multi-layered networks in Chapter 3, We formally define the connectivity optimization problem (OPERA) on the proposed MULAN model for multi-layered networks as follows.

Problem 10. Opera on MuLaN

Given: *(1) a multi-layered network* $\Gamma =< \mathbf{G}, \mathcal{A}, \mathcal{D}, \theta, \varphi >$*; (2) a control layer* $\mathbf{A}_l$*; (3) an impact function* $\mathbb{I}(.)$*; and (4) an integer* k *as operation budget;*

Output: *a set of* k *nodes* $\mathcal{S}_l$ *from the control layer (*$\mathbf{A}_l$*) such that* $\mathbb{I}(\mathcal{S}_l)$ *(the overall impact of* $\mathcal{S}_l$*) is maximized.*

In the above definition, the control layer $\mathbf{A}_l$ indicates the sources of the 'attack'; and the $g \times 1$ vector α indicates the target layer(s) as well as their relative weights. For instance, in Figure 3.1(a), we can choose layer-1 as the control layer (indicated by the strike sign); and set $\alpha = [0\ 1\ 0\ 1\ ]'$, which means that both layer-2 and layer-4 are the target layers (indicated by the star signs) with equal weights between them. In this

[7] In this section, we use the adjacency matrix $\mathbf{A}_i$ to represent the corresponding network. Thus, the $\mathbf{A}_i$ here is equivalent to the G_i in Section 3.

Table 5.3: Main Symbols for MuLaN.

Symbol	Definition and Description
$\mathbf{A}, \mathbf{B}$	the adjacency matrices (bold upper case)
$\mathbf{a}, \mathbf{b}$	column vectors (bold lower case)
$\mathcal{A}, \mathcal{B}$	sets (calligraphic)
$\mathbf{A}(i,j)$	the element at i^{th} row j^{th} column in 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 $\mathbf{A}$
$\mathbf{G}$	the layer-layer dependency matrix
$\mathcal{A}$	adjacency matrices at each layers $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_g\}$
$\mathcal{D}$	cross-layer node-node dependency matrices
θ, φ	one to one mapping functions
Γ	multi-layered network MuLaN $\Gamma =< \mathbf{G}, \mathcal{A}, \mathcal{D}, \theta, \varphi >$
$\mathcal{S}_i, \mathcal{T}_i, \ldots$	node sets in layer $\mathbf{A}_i$(calligraphic)
$\mathcal{S}_{i\to j}$	nodes in $\mathbf{A}_j$ that depend on nodes $\mathcal{S}$ in $\mathbf{A}_i$
$\mathcal{N}(\mathcal{S}_i)$	nodes and cross-layer links that depend on $\mathcal{S}_i$
m_i, n_i	number of edges and nodes in layer $\mathbf{A}_i$
$\lambda_{<\mathbf{A},j>}, \mathbf{u}_{<\mathbf{A},j>}$	j^{th} largest eigenvalue (in module) and eigenvector $\mathbf{A}$
$\lambda_{\mathbf{A}}, \mathbf{u}_{\mathbf{A}}$	first eigenvalue and eigenvector of network $\mathbf{A}$
$C(\mathbf{A}_i, f)$	connectivity of network $\mathbf{A}_i$ under mapping function f
$I_{\mathbf{A}}(\mathcal{S}_i)$	impact of node set $\mathcal{S}_i$ on network $\mathbf{A}$
$\mathbb{I}(\mathcal{S}_i)$	overall impact of node set $\mathcal{S}_i$ on MuLaN

example, once a subset of nodes $\mathcal{S}$ in layer-1 are attacked/deleted (e.g., shaded circle nodes), all the nodes from layer-2 and layer-3 that are dependent on $\mathcal{S}$ (e.g., shaded parallelogram and triangle nodes) will be disabled/deleted, which will in turn cause the disfunction of the nodes in layer-4 (e.g., shaded diamond nodes) that depend on the affected nodes in layer-2 or layer-3. Our goal is to choose k nodes from layer-1 that have the maximal impact on both layer-2 and layer-4, i.e., to simultaneously decrease the connectivity $C(\mathbf{A}_2, f)$ and $C(\mathbf{A}_4, f)$ as much as possible.

5.2.2 *Theoretical Analysis*

In this section, we present the major theoretical results of the connectivity optimization problem (OPERA) on multi-layered networks defined in Problem 1. It says that for **any** connectivity function $C(\mathbf{A}, f)$ in the SUBLINE family (Eq. (3.6)), for **any** multi-layered network in the MULAN family (Definition 1), the connectivity optimization problem (OPERA, Problem 1) bears **diminishing returns property**.

Let us start with the base case, where there is only one single input network. In this case, $\Gamma =< \mathbf{G}, \mathcal{A}, \mathcal{D}, \theta, \varphi >$ in Problem 1 degenerates to a single-layered network $\mathbf{A}$, which is both the control layer as well as the sole control target (i.e., $\alpha = 1$, and $l = 1$). With such a setting, Lemma 15 says that OPERA enjoys the *diminishing returns property*, that is, the overall impact function $\mathbb{I}(\mathcal{S}_1)$ (which in this case degenerates to $I(\mathcal{S})$, i.e., the impact of the node set $\mathcal{S}$ on network G itself) is (a) monotonically non-decreasing; (b) sub-modular; and (c) normalized.

Lemma 15. Diminishing Returns Property of a Single-layered Network. *Given a simple undirected, un-weighted network* $\mathbf{A}$*, for any connectivity function* $C(\mathbf{A}, f)$ *in the* SUBLINE *family, the impact function* $I(\mathcal{S})$ *is (a) monotonically non-decreasing; (b) sub-modular; and (c) normalized, where* $\mathcal{S} \subseteq \mathbf{A}$.

Proof. By the definition of the connectivity function $C(\mathbf{A}, f)$ (Eq. (3.6)), we have

$$I(\mathcal{S}) = \sum_{\pi \subseteq \mathbf{A}} f(\pi) - \sum_{\pi \subseteq \mathbf{A} \setminus \mathcal{S}} f(\pi) = \sum_{\pi \subseteq \mathbf{A},\ \pi \cap \mathcal{S} \neq \Phi} f(\pi)$$

where Φ is the empty set. Apparently, we have $I(\Phi) = 0$ since $f(\Phi) = 0$. In other words, the impact function $I(\mathcal{S})$ is normalized.

Let $\mathcal{I}, \mathcal{J}, \mathcal{K}$ be three sets and $\mathcal{I} \subseteq \mathcal{J}$. We further define three sets as follows: $\mathcal{S} = \mathcal{I} \cup \mathcal{K}$, $\mathcal{T} = \mathcal{J} \cup \mathcal{K}$, $\mathcal{R} = \mathcal{J} \setminus \mathcal{I}$.

We have

$$\begin{aligned} I(\mathcal{J}) - I(\mathcal{I}) &= \sum_{\pi \subseteq \mathbf{A},\ \pi \cap \mathcal{J} \neq \Phi} f(\pi) - \sum_{\pi \subseteq \mathbf{A},\ \pi \cap \mathcal{I} \neq \Phi} f(\pi) \\ &= \sum_{\pi \subseteq \mathbf{A},\ \pi \cap (\mathcal{J} \setminus \mathcal{I}) \neq \Phi} f(\pi) = \sum_{\pi \subseteq \mathbf{A},\ \pi \cap \mathcal{R} \neq \Phi} f(\pi) \\ &\geq 0 \end{aligned}$$

which proves the monotonicity of the impact function $I(\mathcal{S})$.

Let us define another set $\mathcal{P} = \mathcal{T} \setminus \mathcal{S}$. We have that $\mathcal{P} = (\mathcal{J} \cup \mathcal{K}) \setminus (\mathcal{I} \cup \mathcal{K}) = \mathcal{R} \setminus (\mathcal{R} \cap \mathcal{K}) \subseteq \mathcal{R} = \mathcal{J} \setminus \mathcal{I}$. Then, we have

$$I(\mathcal{T}) - I(\mathcal{S}) = \sum_{\pi \subseteq \mathbf{A},\ \pi \cap \mathcal{P} \neq \Phi} f(\pi) \leq I(\mathcal{J}) - I(\mathcal{I})$$

which completes the proof of the sub-modularity of the impact function $I(\mathcal{S})$. □

In order to generalize Lemma 15 to an arbitrary, generic member in the MuLaN family, we first need the following lemma, which says that the set-ordering relationship in a supporting layer is preserved through dependency links in all dependent layers of MuLaN.

Lemma 16. Set-ordering Preservation Lemma on DAG. *Given a multi-layered network* $\Gamma =< \mathbf{G}, \mathcal{A}, \mathcal{D}, \theta, \varphi >$, *and the dependency network* $\mathbf{G}$ *is a directed acyclic graph (DAG). For two node sets* $\mathcal{I}_l, \mathcal{J}_l$ *in* $\mathbf{A}_l$ *such that* $\mathcal{I}_l \subseteq \mathcal{J}_l$, *we have that in any layer* $\mathbf{A}_i$ *in the system,* $\mathcal{I}_{l\to i} \subseteq \mathcal{J}_{l\to i}$ *holds, where* $\mathcal{I}_{l\to i}$ *and* $\mathcal{J}_{l\to i}$ *are the node sets in layer* $\mathbf{A}_i$ *that depend on* $\mathcal{I}_l$ *and* $\mathcal{J}_l$ *in layer* $\mathbf{A}_l$ *respectively.*

Proof. If $l = i$, we have $\mathcal{J}_{l\to i} = \mathcal{J} \subseteq \mathcal{I}_{l\to i} = \mathcal{I}$ and Lemma 16 holds.

Second, if layer-i does not depend on layer-l either directly or indirectly, we have $\mathcal{J}_{l\to i} = \mathcal{I}_{l\to i} = \Phi$, where Φ is an empty set. Lemma 16 also holds.

If layer-i does depend on layer-l through the layer-layer dependency network $\mathbf{G}$, we will prove Lemma 16 by induction. Let $\text{len}(l \rightsquigarrow i)$ be the maximum length of the path from node l to node i on the layer-layer dependency network $\mathbf{G}$. Since $\mathbf{G}$ is a DAG, we have that $\text{len}(l \rightsquigarrow i)$ is a finite number.

Base Case. Suppose $\text{len}(l \rightsquigarrow i) = 1$, we have that layer-$i$ directly depends on layer-l. Let $\mathcal{R}_l = \mathcal{J}_l \setminus \mathcal{I}_l$. We have that

$$\mathcal{J}_{l\to i} = \mathcal{I}_{l\to i} \cup \mathcal{R}_{l\to i} \supseteq \mathcal{I}_{l\to i} \tag{5.19}$$

which complete the proof for the base case where $\text{len}(l \rightsquigarrow i) = 1$.

Induction Step. Suppose Lemma 16 holds for $\text{len}(l \rightsquigarrow i) \leq q$, where q is a positive integer. We will prove that Lemma 16 also holds for $\text{len}(l \rightsquigarrow i) = q + 1$.

Suppose layer-i directly depends on layer-i_x ($x = 1, ..., d(i)$, where $d(i)$ is the in-degree of node i on $\mathbf{G}$). Since $\mathbf{G}$ is a DAG, we have that $\text{len}(l \rightsquigarrow i_x) \leq q$. By the induction hypothesis, given $\mathcal{I}_l \subseteq \mathcal{J}_l$, we have that $\mathcal{I}_{l\to i_x} \subseteq \mathcal{J}_{l\to i_x}$.

We further have $\mathcal{I}_{l\to i} = \cup_{x=1,\ldots,d(i)} \ \ (\mathcal{I}_{l\to i_x})_{i_x\to i}$.

Let $\mathcal{R}_{l\to i_x} = \mathcal{J}_{l\to i_x} \setminus \mathcal{I}_{l\to i_x}$ for $x = 1, \ldots, d(i)$. We have that

$$\begin{aligned}\mathcal{J}_{l\to i} &= [\cup_{x=1,\ldots,d(i)} \ \ (\mathcal{I}_{l\to i_x})_{i_x\to i}] \\ &\cup [\cup_{x=1,\ldots,d(i)} \ \ (\mathcal{R}_{l\to i_x})_{i_x\to i}] \\ &= \mathcal{I}_{l\to i} \cup \mathcal{R}_{l\to i} \supseteq \mathcal{I}_{l\to i}\end{aligned} \tag{5.20}$$

which completes the proof of the induction step.

Putting everything together, we have completed the proof for Lemma 16. □

Notice that in the proof of Lemma 16, it requires the layer-layer dependency network $\mathbf{G}$ to be a DAG so that the longest path from the control layer $\mathbf{A}_l$ to any target layer $\mathbf{A}_t$ is of finite length. To further generalize it to arbitrary dependency structures, we need the following lemma, which says that the dependent paths from control layer to target layer in any arbitrarily structured dependency network can be reduced to a DAG.

Lemma 17. DAG Dependency Reduction Lemma. *Given a multi-layered network $\Gamma =< \mathbf{G}, \mathcal{A}, \mathcal{D}, \theta, \varphi >$ with arbitrarily structured layer-layer dependency network $\mathbf{G}$, a control layer $\mathbf{A}_l$, and a target layer $\mathbf{A}_t$, the dependent paths constructed by Algorithm 10 can be reduced to a DAG.*

Proof. In Algorithm 10, Tarjan Algorithm is first used to find out all strongly connected components $\mathcal{V} = \{\mathcal{SC}_1, \mathcal{SC}_2, \ldots, \mathcal{SC}_f\}$ in layer-layer dependency network $\mathbf{G}$. The cross-component dependency edges are denoted as $\mathcal{E} = \{E_{i,j}\}_{i,j=1,\ldots,f,i\neq j}$ where $\langle u, v\rangle \in E_{i,j}$ iff $\mathbf{G}(u, v) = 1$ and $\mathbf{A}_u \in \mathcal{SC}_i$, $\mathbf{A}_v \in \mathcal{SC}_j$. Based on the node set $\mathcal{V}$ and the edge set $\mathcal{E}$, a directed meta-graph $\mathcal{G}$ can be constructed where $\mathcal{G}(u, v) = 1$ iff $E_{i,j} \neq \phi$. The meta-graph $\mathcal{G}$ is acyclic. Otherwise, the cycle in $\mathcal{G}$ would be merged into a large strongly connected component by Tarjan Algorithm in the first place. Suppose the control layer $\mathbf{A}_l$ and the target layer $\mathbf{A}_t$ are located in strongly connected component $\mathcal{SC}_i$ and $\mathcal{SC}_j$ respectively, then a set of acyclic paths $\mathcal{P}$ from $\mathcal{SC}_i$ and $\mathcal{SC}_j$ can be extracted from $\mathcal{G}$. To show that the dependent paths from $\mathbf{A}_l$ to $\mathbf{A}_t$ is DAG, we only need to show that each meta-path $\mathbf{P} \in \mathcal{P}$ can be unfolded into a DAG.

Here we proceed to show how a meta-path $\mathbf{P}$ can be represented with a DAG. As the nodes in $\mathbf{P}$ are strongly connected components that contain cycles, and the edges in $\mathbf{P}$ contain corresponding cross-component edges that would not introduce any cycles, representing $\mathbf{P}$ with a DAG can be converted to a problem of unfolding the cyclic dependent paths in a strongly connected component into a DAG. As described in Algorithm 13, a strongly connected component $\mathcal{Q}$ can be partitioned into two parts: (1) a DAG that contains all acyclic links (denoted as $\mathbf{G}_{\mathcal{Q},0}$) and (2) links that enclose cycles in $\mathcal{Q}$ (denoted as $E_{\mathcal{Q},0}$). Therefore, given a strongly connected component $\mathcal{Q}$ and a set of dependent nodes $\{\mathcal{T}_v\}_{\mathbf{A}_v \in \mathbf{Q}}$ in $\mathcal{Q}$, the dependent cycle can be replaced by a chain of $\mathbf{G}_{\mathcal{Q},0}$'s replicas, where the two adjacent replicas are linked by $E_{\mathcal{Q},0}$ until the number of the dependent nodes in the connected component converges (step 5 to 23 in Algorithm 12). As the number of dependent nodes keeps increasing in each iteration and is upper bounded by the total number of nodes in $\mathcal{Q}$, the repetition is guaranteed to stop at a stable state within finite iterations. Since $\mathbf{G}_{\mathcal{Q},0}$ is a DAG, the links ($E_{\mathcal{Q},0}$) between each replicas $\{\mathbf{G}_{\mathcal{Q},1}, \ldots, \mathbf{G}_{\mathcal{Q},L}\}$ would not introduce any cycle, the resulting graph $\mathbf{G}_{\mathcal{Q}}$ is also a DAG. Therefore, the dependent paths constructed by Algorithm 10 from $\mathbf{A}_l$ and $\mathbf{A}_t$ can be represented as a DAG. $\square$

A complete DAG reduction algorithm is summarized from Algorithm 10 to 13.

In Algorithm 10, step 1 runs Tarjan Algorithm Tarjan (1972) to find out all the strongly connected components in layer-layer dependency network $\mathbf{G}$. Step 2 collects all cross-component edges into set $\mathcal{E}$. In the following step, a meta-graph $\mathcal{G}$ is constructed based on $\mathcal{V}$ and $\mathcal{E}$. In step 4 and 5, the connected components that contain control layer and target layer are located ($\mathcal{SC}_i$ and $\mathcal{SC}_j$). Step 6 finds out all meta-paths from $\mathcal{SC}_i$ to $\mathcal{SC}_j$. In step 7, the final DAG $\mathbf{G}_D$ and dependent node set $\mathcal{S}_{l\to t}$ are initialized as empty sets. From step 8 to step 11, the DAG $\mathbf{G}_D^{\mathbf{P}}$ and dependent node set $\mathcal{S}_{l\to t}^{\mathbf{P}}$ for each path $\mathbf{P}$ in $\mathcal{P}$ are calculated by function $unfoldPath$, and are used to update $\mathbf{G}_D$ and $\mathcal{S}_{l\to t}$ in step 10.

To illustrate how Algorithm 10 works, we present a simple example in Figure 5.14. In the example, the dependency network $\mathbf{G}$ contains three layers, where $\mathbf{A}_1$ is the control layer and $\mathbf{A}_3$ is the target layer. Specifically, $\mathbf{A}_2$ is a dependent layer of $\mathbf{A}_1$; while $\mathbf{A}_2$ and $\mathbf{A}_3$ are inter-dependent to each other. The toy example has two strongly

[8] A widely used strongly connect component detection algorithm in Tarjan (1972).

Algorithm 10 DAG Reduction Algorithm

Input: (1) A multi-layered network Γ, (2) a control layer $\mathbf{A}_l$, (3) a set of node $\mathcal{S}_l$ in layer $\mathbf{A}_l$ and (4) a target layer $\mathbf{A}_t$

Output: (1) a DAG $\mathbf{G}_D$ that contains all the dependent paths from $\mathcal{S}_l$ in layer $\mathbf{A}_l$ to $\mathbf{A}_t$ and (2) $\mathcal{S}_{l\to t}$.

find out all strongly connected components in $\mathbf{G}$ as $\mathcal{V} \leftarrow \{\mathcal{SC}_1, \mathcal{SC}_2, \ldots, \mathcal{SC}_f\}$ with Tarjan Algorithm[8]

set $\mathcal{E} \leftarrow \{E_{i,j}\}_{i,j=1,\ldots,f}$, where $\langle u, v\rangle \in E_{i,j}$ iff $\mathbf{G}(u,v) = 1$ and $\mathbf{A}_u \in \mathcal{SC}_i$, $\mathbf{A}_v \in \mathcal{SC}_j$

construct meta-graph $\mathcal{G}$ from $\mathcal{V}$ s.t. $\mathcal{G}(i,j) = 1$ iff $E_{i,j} \neq \phi$

$\mathcal{SC}_i \leftarrow$ connected component that contains $\mathbf{A}_l$

$\mathcal{SC}_j \leftarrow$ connected component that contains $\mathbf{A}_t$

find out all paths $\mathcal{P}$ from $\mathcal{SC}_i$ to $\mathcal{SC}_j$ in $\mathcal{G}$

initialize $\mathbf{G}_D \leftarrow \phi$, $\mathcal{S}_{l\to t} = \phi$

for each path $\mathbf{P}$ in $\mathcal{P}$ **do**

  $[\mathbf{G}_D^{\mathbf{P}}, \mathcal{S}_{l\to t}^{\mathbf{P}}] \leftarrow unfoldPath(\mathbf{P}, \mathcal{G}, \mathcal{S}_l, \Gamma, \mathcal{V}, \mathcal{E})$

  $\mathbf{G}_D \leftarrow \mathbf{G}_D \cup \mathbf{G}_D^{\mathbf{P}}$, $\mathcal{S}_{l\to t} \leftarrow \mathcal{S}_{l\to t} \cup \mathcal{S}_{l\to t}^{\mathbf{P}}$

end for

return $[\mathbf{G}_D, \mathcal{S}_{l\to t}]$

connected components $\{\mathcal{SC}_1, \mathcal{SC}_2\}$ and one cross-component edge set $E_{1,2} = \{\langle 1, 2\rangle\}$. The meta-graph $\mathcal{G}$ is a link graph with just two nodes.

Algorithm 11 UnfoldPath: Construct DAG from meta-path

Input: (1) A meta-path $\mathbf{P} = \mathcal{SC}_i \to \ldots \to \mathcal{SC}_j$, (2) a meta-graph $\mathcal{G}$, (3) a set of nodes $\mathcal{S}_l$ in $\mathbf{A}_l \in \mathcal{SC}_i$, (4) a multi-layered network Γ, (5) all strongly connected components $\mathcal{V}$ and (6) all cross-component edges $\mathcal{E}$

Output: (1) a DAG $\mathbf{G}_D^{\mathbf{P}}$ and (2) $\mathcal{S}_{l\to t}^{\mathbf{P}}$.

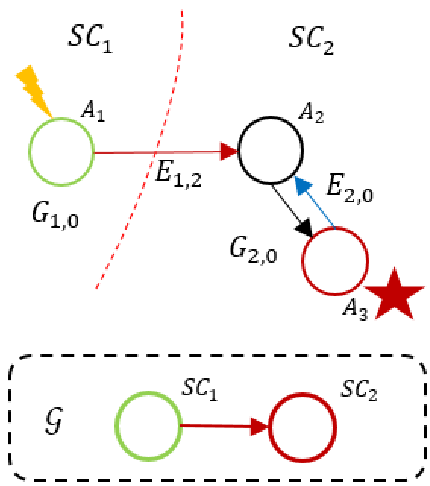


Figure 5.14: A Cyclic Dependency Multi-layered Network.

append ϕ to the end of meta-path $\mathbf{P}$

set $\mathcal{Q} = \mathcal{SC}_i$

$i_q \leftarrow$ index of connected component $\mathcal{Q}$ in meta-graph $\mathcal{G}$

$i''_q \leftarrow -1$

for each layer $\mathbf{A}_v$ in $\mathcal{Q}$ **do**

  initialize $\mathcal{T}_v \leftarrow \phi$

end for

$\mathcal{T}_l \leftarrow \mathcal{T}_l \cup \mathcal{S}_l$

set root $\mathbf{R} \leftarrow \mathbf{A}_l$

while true do

  $[\mathbf{G}^{\mathbf{P}}_{\mathcal{Q}}, \{\mathcal{S}^{\mathbf{P}}_{l \to v}\}_{\mathbf{A}_v \in \mathcal{Q}}] \leftarrow unfoldSC(\mathcal{Q}, \{\mathcal{T}_v\}_{\mathbf{A}_v \in \mathcal{Q}}, \mathbf{R})$

  if $i''_q = -1$ **then**

    $\mathbf{G}^{\mathbf{P}}_{D} \leftarrow \mathbf{G}^{\mathbf{P}}_{\mathcal{Q}}$

  else

    for each $\langle u, v \rangle \in E_{i''_q, i_q}$ **do**

      link layer $\mathbf{A}_u^{L_{i''_q}} \in \mathbf{G}^{\mathbf{P}}_{D}$ to layers $\{\mathbf{A}^{x}_{v}\}_{x=1,\ldots,L_{i_q}} \in \mathbf{G}^{\mathbf{P}}_{\mathcal{Q}}$

17: **end for**
18: **end if**
19: $\mathcal{Q}' \leftarrow \mathcal{Q}.successor()$
20: **if** $\mathcal{Q}' = \phi$ **then**
21: **break**
22: **else**
23: $i'_q \leftarrow$ index of $\mathcal{Q}'$ in meta-graph $\mathcal{G}$
24: **for** each layer $\mathbf{A}_v$ in $\mathcal{Q}'$ **do**
25: initialize $\mathcal{T}_v \leftarrow \phi$
26: **end for**
27: **for** each edge $\langle u, v\rangle \in E_{i_q, i'_q}$ **do**
28: $\mathcal{T}_v \leftarrow \mathcal{T}_v \cup (\mathcal{S}^{\mathbf{P}}_{l \to u})_{u \to v}$
29: **end for**
30: $\mathbf{R} \leftarrow \mathbf{A}_r$, where $\mathbf{A}_r$ is a randomly picked layer from $\mathcal{Q}'$ with $\mathcal{T}_r \neq \phi$
31: $\mathcal{Q} \leftarrow \mathcal{Q}'$
32: $i''_q \leftarrow i_q$
33: $i_q \leftarrow i'_q$
34: **end if**
35: **end while**
36: return $\mathbf{G}^{\mathbf{P}}_D$, $\mathcal{S}^{\mathbf{P}}_{l \to t}$

In Algorithm 11, the first connected component $\mathcal{Q}$ is initialized as the connected component that contains control layer $\mathbf{A}_l$ in step 2, the dependent nodes are initialized as $\mathcal{S}_l$ from step 5 to 8 and the root layer $\mathbf{R}$ is initialized as the control layer $\mathbf{A}_l$. From step 10 to 36, the DAG $\mathbf{G}^{\mathbf{P}}_{\mathcal{Q}}$ and the final dependent nodes in $\mathcal{Q}$ are calculated by function $unfoldSC$ in step 11; $\mathbf{G}^{\mathbf{P}}_{\mathcal{Q}}$ is then added to the final DAG $\mathbf{G}^{\mathbf{P}}_D$ via cross-component links $E_{i''_q, i_q}$ from step 15 to 17. The initial dependent nodes for the next connected component $\mathcal{SC}_{i'_q}$ are computed through cross-component links E_{i_q,i'_q} from step 27 to 29. Step 30 is used to pick a root layer with non-empty dependent node set for $\mathcal{SC}_{i'_q}$.

Algorithm 12 UnfoldSC: Construct DAG from strongly connected component

Input: (1) A strongly connected component $\mathcal{Q}$, (2) a set of initial nodes for each layer $\{\mathcal{T}_v\}_{\mathbf{A}_v \in \mathbf{Q}}$, (3) a root layer $\mathbf{R}$

Output: (1) a DAG $\mathbf{G}_{\mathcal{Q}}$ (2) $\{\mathcal{S}_{l \to v}\}_{\mathbf{A}_v \in \mathcal{Q}}$.

extract DAG and cycle edges $[\mathbf{G}_{\mathcal{Q},0}, E_{\mathcal{Q},0}] \leftarrow \mathit{extractDAG}(\mathcal{Q}, \mathbf{R})$
set $\mathbf{G}_{\mathcal{Q},1} \leftarrow \mathbf{G}_{\mathcal{Q},0}$, denote the layers in $\mathbf{G}_{\mathcal{Q},1}$ as $\{\mathbf{A}_v^1\}$
set $c \leftarrow 1$
initialize $\mathbf{G}_{\mathcal{Q}} \leftarrow \mathbf{G}_{\mathcal{Q},1}$
while true do
  $\{\mathcal{T}_v^c\}_{\mathbf{A}_v \in \mathcal{Q}} \leftarrow$ dependents of $\{\mathcal{T}_v\}_{\mathbf{A}_v \in \mathcal{Q}}$ in $\mathbf{G}_{\mathcal{Q},c}$
  update $\{\mathcal{T}_v\}_{\mathbf{A}_v \in \mathcal{Q}} \leftarrow \{\mathcal{T}_v \cup \mathcal{T}_v^c\}_{\mathbf{A}_v \in \mathcal{Q}}$
  set $\mathbf{G}_{\mathcal{Q},c+1} \leftarrow \mathbf{G}_{\mathcal{Q},0}$, layers in $\mathbf{G}_{\mathcal{Q},c+1}$ are denoted as $\{\mathbf{A}_v^{c+1}\}$
  extend $\mathbf{G}_{\mathcal{Q}} \leftarrow \mathbf{G}_{\mathcal{Q}} \cup \mathbf{G}_{\mathcal{Q},c+1}$
  for each edge $\langle u, v \rangle \in E_{\mathcal{Q},0}$ **do**
    $\mathcal{T}_{u \to v} \leftarrow$ all dependents of $\mathcal{T}_u$ in layer $\mathbf{A}_v$
    if $\mathcal{T}_{u \to v} \nsubseteq \mathcal{T}_v$ **then**
      add edge $\langle \mathbf{A}_u^c, \mathbf{A}_u^{c+1} \rangle$ to $\mathbf{G}_{\mathcal{Q}}$
      update $\mathcal{T}_v \leftarrow \mathcal{T}_v \cup \mathcal{T}_{u \to v}$
    end if
  end for
  if no edge added between $\mathbf{G}_{\mathcal{Q},c}$ and $\mathbf{G}_{\mathcal{Q},c+1}$ **then**
    remove $\mathbf{G}_{\mathcal{Q},c+1}$ from $\mathbf{G}_{\mathcal{Q}}$
    break

20: **else**

21: $c \leftarrow c + 1$

22: **end if**

23: **end while**

24: return $[\mathbf{G}_{\mathcal{Q}}, \{\mathcal{T}_v\}_{\mathbf{A}_v \in \mathcal{Q}}]$

Algorithm 12 is used to unfold a strongly connected component into a DAG. In step 1, the input connected component $\mathcal{Q}$ is partitioned into a DAG $\mathbf{G}_{\mathcal{Q},0}$ and a set of cycle links $E_{\mathcal{Q},0}$. In step 2, the DAG $\mathbf{G}_{\mathcal{Q}}$ is initialized by $\mathbf{G}_{\mathcal{Q},1}$, which is a replica of $\mathbf{G}_{\mathcal{Q},0}$. From step 5 to 23, the algorithm keeps appending replicas of $\mathbf{G}_{\mathcal{Q},0}$ ($\mathbf{G}_{\mathcal{Q},c+1}$) onto $\mathbf{G}_{\mathcal{Q}}$ (step 8 to 16) until no new nodes are added to the dependent node set $\{\mathcal{T}_v\}_{\mathbf{A}_v \in \mathcal{Q}}$ (step 17-19).

For the example in Figure 5.14, $\mathcal{SC}_1$ is unfolded as $\mathbf{G}_1$ with one node $\mathbf{A}_1^1$ in Figure 5.15. The initial dependent node set $\mathcal{T}_2$ for layer $\mathbf{A}_2$ can be calculated through $E_{1,2}$ as $\mathcal{T}_{1\to 2}$. For $\mathcal{SC}_2$, it is first partitioned into a DAG $\mathbf{G}_{2,0}$ and a cycle edge set $E_{2,0} = \{\langle \mathbf{A}_3, \mathbf{A}_2 \rangle\}$ as shown in Figure 5.14. Suppose that the dependent node set in $\mathcal{SC}_2$ converges in L_2 iterations, then the DAG for $\mathcal{SC}_2$ can be presented with L_2 replicas of $\mathbf{G}_{2,0}$ linked by edges $\{\langle \mathbf{A}_3^c, \mathbf{A}_2^{c+1} \rangle\}_{c=1,\dots,L_2-1}$ as shown in Figure 5.15. Putting all together, the final DAG $\mathbf{G}_D$ can be constructed by linking $\mathbf{A}_1^1$ in $\mathbf{G}_1$ with $\{\mathbf{A}_2^x\}_{x=1,\dots,L_2}$ in $\mathbf{G}_2$.

Algorithm 13 is used to partition a strongly connected component $\mathcal{Q}$ into a DAG $\mathbf{G}_{\mathcal{Q}}$ and an edge set $E_{\mathcal{Q},0}$ that contains all cycle edges. The basic idea is to use Breadth-First-Search algorithm to traverse all the edges in the graph. In step 1 and 2, $\mathbf{G}_{\mathcal{Q},0}$ and $E_{\mathcal{Q},0}$ are initialized as $\mathcal{Q}$ and ϕ respectively. For each edge $\langle \mathbf{A}_u, \mathbf{A}_v \rangle$ in $\mathcal{Q}$, if $\mathbf{A}_v$ appears in $\mathbf{A}_u$'s ancestor list $\mathcal{L}_u$, then $\langle \mathbf{A}_u, \mathbf{A}_v \rangle$ would be removed from $\mathbf{G}_{\mathcal{Q},0}$ and added to $E_{\mathcal{Q},0}$ (step11 to 13).

The algorithms used in Lemma 17 together with Lemma 16 guarantee that set-

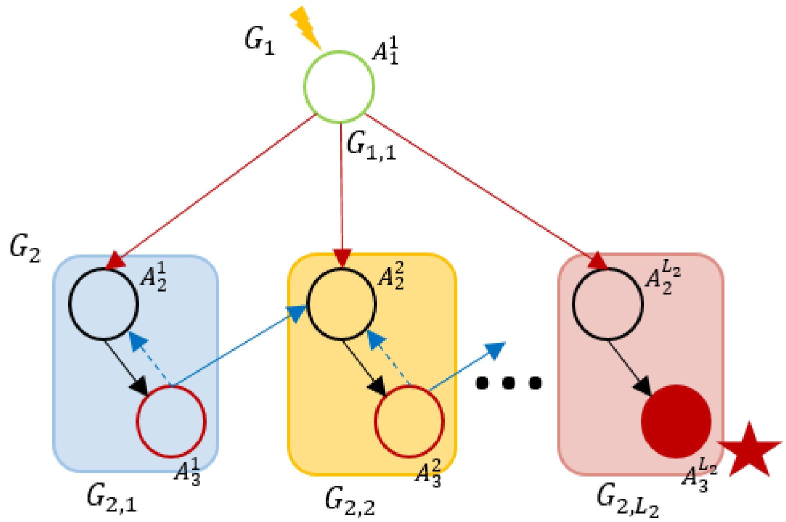


Figure 5.15: Constructed DAG for Figure 5.14.

ordering preservation property also holds in multi-layered networks with arbitrarily structured dependency graph $\mathbf{G}$.

Now, we are ready to present our main theorem as follows.

Theorem 6. Diminishing Returns Property of MuLaN. *For* **any** *connectivity function $C(\mathbf{A})$ in the* SubLine *family (Eq. (3.6)) and* **any** *multi-layered network in the* MuLaN *family (Definition 1); the overall impact of node set $\mathcal{S}_l$ in the control layer l, $\mathbb{I}(\mathcal{S}_l) = \sum_{i=1}^{g} \alpha_i I(\mathcal{S}_{l\to i})$, is (a) monotonically non-decreasing; (b) sub-modular; and (c) normalized.*

Proof. We first prove the sub-modularity of function $\mathbb{I}(\mathcal{S}_l)$. Let $\mathcal{I}_l, \mathcal{J}_l, \mathcal{K}_l$ be three node sets in layer $\mathbf{A}_l$ and $\mathcal{I}_l \subseteq \mathcal{J}_l$. Define the following two sets as: $\mathcal{S}_l = \mathcal{I}_l \cup \mathcal{K}_l$ and $\mathcal{T}_l = \mathcal{J}_l \cup \mathcal{K}_l$. We have that

$$\begin{aligned}\mathbb{I}(\mathcal{S}_l) - \mathbb{I}(\mathcal{I}_l) &= \sum_{i=1}^{g} \alpha_i I(\mathcal{S}_{l\to i}) - \sum_{i=1}^{g} \alpha_i I(\mathcal{I}_{l\to i}) \\ &= \sum_{i=1}^{g} \alpha_i (I(\mathcal{S}_{l\to i}) - I(\mathcal{I}_{l\to i}))\end{aligned} \tag{5.21}$$

Algorithm 13 ExtractDAG: extract DAG from strongly connected component

Input: (1) A strongly connected component $\mathcal{Q}$ and (2) a root layer $\mathbf{R}$ in the connected component

Output: (1) a DAG $\mathbf{G}_{\mathcal{Q},0}$ (2) edge set $E_{\mathcal{Q},0}$ that contains all cycle edges.

initialized $\mathbf{G}_{\mathcal{Q},0} \leftarrow \mathcal{Q}$
initialized $E_{\mathcal{Q},0} \leftarrow \phi$
for each layer $\mathbf{A}_v \in \mathcal{Q}$ **do**
    initialize its ancestor list $\mathcal{L}_v \leftarrow \phi$
end for
initialize a queue $\mathcal{T} \leftarrow \phi$
$\mathcal{T}.enqueue(\mathbf{R})$
while $\mathcal{T} \neq \phi$ **do**
    $\mathbf{A}_u \leftarrow \mathcal{T}.dequeue()$
    for each dependent layer $\mathbf{A}_v$ of $\mathbf{A}$ **do**
        if $\mathbf{A}_v \in \mathcal{L}_u$ **then**
            remove edge $\langle \mathbf{A}_u, \mathbf{A}_v \rangle$ from $\mathbf{G}_{\mathcal{Q},0}$
            $E_{\mathcal{Q},0} \leftarrow E_{\mathcal{Q},0} \cup \langle \mathbf{A}_u, \mathbf{A}_v \rangle$
        else
            $\mathcal{T}.enqueue(\mathbf{A}_v)$
            $\mathcal{L}_v \leftarrow \mathcal{L}_v \cup \mathcal{L}_u \cup \{\mathbf{A}_u\}$
        end if
    end for
end while
return $[\mathbf{G}_{\mathcal{Q},0},\ E_{\mathcal{Q},0}]$

$$\mathbb{I}(\mathcal{T}_l) - \mathbb{I}(\mathcal{J}_l) = \sum_{i=1}^{g} \alpha_i I(\mathcal{T}_{l\to i}) - \sum_{i=1}^{g} \alpha_i I(\mathcal{J}_{l\to i}) \tag{5.22}$$
$$= \sum_{i=1}^{g} \alpha_i (I(\mathcal{T}_{l\to i}) - I(\mathcal{J}_{l\to i}))$$

$\forall i = 1, \ldots, g$, it is obvious that $\mathcal{S}_{l\to i} = \mathcal{I}_{l\to i} \cup \mathcal{K}_{l\to i}$, $\mathcal{T}_{l\to i} = \mathcal{J}_{l\to i} \cup \mathcal{K}_{l\to i}$. By Lemma 16, we have $\mathcal{I}_{l\to i} \subseteq \mathcal{J}_{l\to i}$. Furthermore, by the sub-modularity of $I(\mathcal{S}_i)$ on $\mathbf{A}_i$ (Lemma 15), we have that

$$I(\mathcal{S}_{l\to i}) - I(\mathcal{I}_{l\to i}) \geq I(\mathcal{T}_{l\to i}) - I(\mathcal{J}_{l\to i})$$

Since for $\forall i$, we have $\alpha_i \geq 0$. Therefore

$$\sum_{i=1}^{g} \alpha_i (I(\mathcal{S}_{l\to i}) - I(\mathcal{I}_{l\to i})) \geq \sum_{i=1}^{g} \alpha_i (I(\mathcal{T}_{l\to i}) - I(\mathcal{J}_{l\to i})) \tag{5.23}$$

Putting Eq. (5.21), (5.22) and (5.23) together, we have that

$$\mathbb{I}(\mathcal{S}_l) - \mathbb{I}(\mathcal{I}_l) \geq \mathbb{I}(\mathcal{T}_l) - \mathbb{I}(\mathcal{J}_l)$$

which completes the proof that $\mathbb{I}(\mathcal{S}_l)$ is sub-modular.

Notice that the connectivity function $C(\mathbf{A})$ in the SubLine family is monotonically non-decreasing. By Eq. (5.21), we have that

$$\mathbb{I}(\mathcal{S}_l) - \mathbb{I}(\mathcal{I}_l) = \sum_{i=1}^{g} \alpha_i (C(\mathbf{A}_i \setminus \mathcal{I}_l) - C(\mathbf{A}_i \setminus \mathcal{S}_l)) \geq 0$$

which completes the proof that $\mathbb{I}(\mathcal{S}_l)$ is monotonically non-decreasing.

Finally, notice that for each dependent layer, the impact function $I(\mathcal{S}_i)$ is normalized (Lemma 15); and for $i = 1, \ldots, g$, $\Phi_{l\to i} = \Phi$ (an empty set). Therefore we have that $\mathbb{I}(\Phi) = 0$. In other words, $\mathbb{I}(\mathcal{S}_l)$ is also normalized. □

5.2.3 Proposed Algorithm

In this section, we introduce our algorithm to solve Opera (Problem 1), followed by some analysis in terms of the optimization quality as well as the complexity.

A Generic Solution Framework. Finding out the global optimal solution for Problem 10 by a brute-force method would be computationally intractable, due to the exponential enumeration. Nonetheless, the diminishing returns property of the impact function $\mathbb{I}(.)$ (Theorem 6) immediately lends itself to a greedy algorithm for solving OPERA with any connectivity function in the SUBLINE family and arbitrary member in the MULAN family, as summarized in Algorithm 14.

In Algorithm 14, step 2-4 calculate the impact score $\mathbb{I}(v_0)$ $(v_0 = 1, 2, ...)$ for each node in the control layer $\mathbf{A}_l$. Step 5 selects the node with the maximum impact score. In each iteration in step 7-19, we select one of the remaining $(k-1)$ nodes, which would make the maximum marginal increase in terms of the current impact score (step 12, $\text{margin}(v_0) = \mathbb{I}(\mathcal{S} \cup \{v_0\}) - \mathbb{I}(\mathcal{S})$). In order to further speed up the computation, the algorithm admits an *optional* lazy evaluation strategy (adopted from Leskovec *et al.* (2007c)) by activating an optional 'if' condition in Step 11.

Note that it is easy to extend Algorithm 14 to the scenario where we have multiple control layers. Suppose $\mathcal{A}_l = \{\mathbf{A}_{l_1}, \mathbf{A}_{l_2}, \ldots, \mathbf{A}_{l_x}\}$ is a set of control layers, to select best k nodes from $\mathcal{A}_l$, we only need to scan over all the nodes in $\mathcal{A}_l$ in step 2 and step 9 respectively, and pick the highest impact node from the entire candidate set in step 5 and 18. Consequently, the resulting set $\mathcal{S}$ returned from the algorithm would contain the k highest impact nodes over $\mathcal{A}_l$.

Algorithm Analysis. Here, we analyze the optimality as well as the complexity of Algorithm 14, which are summarized in Lemma 18-20. According to these lemmas, our proposed Algorithm 1 leads to a *near-optimal* solution with *linear* complexity.

Lemma 18. Near-optimality. *Let $\mathcal{S}_l$ and $\tilde{\mathcal{S}}_l$ be the sets selected by Algorithm 14 and the brute-force algorithm, respectively. Let $\mathbb{I}(\mathcal{S}_l)$ and $\mathbb{I}(\tilde{\mathcal{S}}_l)$ be the overall impact of $\mathcal{S}_l$ and $\tilde{\mathcal{S}}_l$. Then $\mathbb{I}(\mathcal{S}_l) \geq (1 - 1/e)\mathbb{I}(\tilde{\mathcal{S}}_l)$.*

Algorithm 14 Opera: A Generic Solution Framework

Input: (1) A multi-layered network Γ, (2) a control layer $\mathbf{A}_l$, (3) an overall impact function $\mathbb{I}(\mathcal{S}_l)$ and (4) an integer k

Output: a set of k nodes $\mathcal{S}$ from the control layer $\mathbf{A}_l$.

initialize $\mathcal{S}$ to be empty
for each node v_0 in layer $\mathbf{A}_l$ **do**
  calculate margin$(v_0) \leftarrow \mathbb{I}(v_0)$
end for
find $v = \text{argmax}_{v_0}\text{margin}(v_0)$ and add v to $\mathcal{S}$
set margin$(v) \leftarrow -1$
for $i = 2$ to k **do**
  set maxMargin $\leftarrow -1$
  for each node v_0 in layer $\mathbf{A}_l$ **do**
    `/*an optional 'if' for lazy eval.*/`
    if margin$(v_0) >$ maxMargin **then**
      calculate margin$(v_0) \leftarrow \mathbb{I}(\mathcal{S} \cup \{v_0\}) - \mathbb{I}(\mathcal{S})$
      if margin$(v_0) >$ maxMargin **then**
        set maxMargin $\leftarrow$ margin(v_0) and $v \leftarrow v_0$
      end if
    end if
  end for
  add v to $\mathcal{S}$ and set margin$(v) \leftarrow -1$
end for
return $\mathcal{S}$

Proof. As proved in Theorem 6, the overall impact function $\mathbb{I}(\mathcal{S})$ ($\mathcal{S} \subseteq \mathbf{A}_l$) is monotonic, sub-modular and normalized. Using the property of such functions in Nemhauser *et al.* (1978), we have $\mathbb{I}(\mathcal{S}_l) \geq (1 - 1/e)\mathbb{I}(\tilde{\mathcal{S}}_l)$.

$\square$

Lemma 19. Time complexity. *Let $h(n_i, m_i, |\mathcal{S}_{l\to i}|)$ be the time to compute the impact of node set $\mathcal{S}_l$ on layer i. The time complexity for selecting $\mathcal{S}$ of size k from the control layer $\mathbf{A}_l$ is upper bounded by $O(k(|\mathcal{N}(\mathbf{A}_l)| + n_l \sum_{i=1}^{g} h(n_i, m_i, |\mathcal{S}_{l\to i}|)))$ where $\mathcal{N}(\mathbf{A}_l)$ denotes the nodes and cross-layer links in Γ that depends on $\mathbf{A}_l$.*

Proof. The greedy algorithm with lazy evaluation strategy needs to iterate over all the nodes in layer $\mathbf{A}_l$ for k time. At each time, the worst case is that we need to evaluate the marginal increase for all unselected nodes in $\mathbf{A}_l$. The overall complexity of finding dependents of every nodes in $\mathbf{A}_l$ is equal to the size of the sub-system that rooted on $\mathbf{A}_l$, which is $|\mathcal{N}(\mathbf{A}_l)|$. And for each unselected node, finding out its current impact to the system as shown in step 3 and step 12 can be upper bounded by the complexity of $\sum_{i=1}^{g} h(n_i, m_i, n_i) + g$ because there are at most g non-zero weighted layers that depend on $\mathbf{A}_l$. Taking these all together, the complexity of selecting set $\mathcal{S}$ from $\mathbf{A}_l$ with Algorithm 14 is $O(k[|\mathcal{N}(\mathbf{A}_l)| + n_l \sum_{i=1}^{g} h(n_i, m_i, |\mathcal{S}_{l\to i}|)])$, where $|\mathcal{N}(\mathbf{A}_l)|$ is upper bounded by $N + L$, which is the sum of total number of nodes and total number of dependency links in Γ. If given that function h is linear to n_i, m_i and $|\mathcal{S}_{l\to i}|$, as $|\mathcal{S}_{l\to i}|$ is upper bounded by n_i, and n_l can be viewed as a constant compared to N, M and L, it is easy to see that the complexity of the algorithm is linear to N, M and L. $\square$

Remarks. Lemma 19 implies a linear time complexity of the proposed Opera algorithm w.r.t. the size of the entire multi-layered network $(N + M + L)$, where N, M, L are the total number of nodes, the total number of within-layer links and the total number of cross-layer links in Γ under the condition that the function h is linear w.r.t. n_i, m_i and $|\mathcal{S}_{l\to i}|$. This condition holds for most of the network connectivity measures in the SubLine family, e.g., the path capacity, the truncated loop capacity, and the triangle capacity. To see this, let us take the most expensive truncated loop capacity as an example. The time complexity for calculating truncated loop capacity in a single network is $O(mr + nr^2)$, where r is the number of eigenvalues used in the calculation and it is often orders of magnitude smaller compared with m and n. On the other hand, we have $|\mathcal{N}(\mathbf{A}_l)| \leq N + L$. Therefore, the overall time complexity for selecting set $\mathcal{S}$ of size k from control layer $\mathbf{A}_l$ is upper bounded by $O(k(N + L + n_l \sum_{i=1}^{g}(m_i r + n_i r^2))) = O(k(N + L + n_l(rM + r^2 N))) = O(k(L + n_l(rM + r^2 N)))$.

Lemma 20. Space complexity. *Let $w(n_i, m_i, |\mathcal{S}_{l\to i}|)$ be a function of n_i, m_i and $|\mathcal{S}_{l\to i}|$ that denotes the space cost to compute $I(\mathcal{S}_{l\to i})$. The space complexity of Algorithm 14 is $O(N + M + L)$ under the condition that the function w is linear w.r.t. n_i, m_i and $|\mathcal{S}_{l\to i}|$.*

Proof. As defined in 19, N, M, L are the total number of nodes, total number of within-layer links and total number of cross-layer links in Γ. Then storing multi-layered network Γ would take a space of $O(N + M + L)$. In Algorithm 14, it takes $O(n_l)$ to save the marginal increase vector (*margin*) and $O(k)$ to save result $\mathcal{S}$. As space for computing $I(\mathcal{S}_{l\to i})$ can be reused for each layer i, then computing $\mathbb{I}(\mathcal{S}_{l\to i})$ is bounded by $\arg max_i w(n_i, m_i, |\mathcal{S}_{l\to i}|)$. If function w is linear w.r.t. n_i, m_i and $|\mathcal{S}_{l\to i}|$, then the space complexity of Algorithm 14 is of $O(N + M + L + k + n_l) + O(\arg max_i(n_i)) + O(\arg max_i(m_i)) = O(N + M + L)$.

□

Remarks. The condition that the function w is linear w.r.t. n_i, m_i and $|\mathcal{S}_{l\to i}|$ holds for most of the network connectivity measures in the SubLine family, which in turn implies a linear space complexity for the proposed Opera algorithm. Again, let us take the truncated loop capacity connectivity as an example. Storing the input MuLaN (Γ) takes $O(N+M+L)$ in space. The space cost to calculate the truncated loop capacity in a single-layered network is $O(m+nr)$, where r is the number of eigenvalues used for the computation. Again, r is usually a much smaller number compared with m and n, and thus is considered as a constant. Therefore, the overall space complexity for Opera with the truncated loop capacity is $O(N+M+L)$.

5.2.4 Experimental Evaluation

In this section, we empirically evaluate the proposed Opera algorithms. All experiments are designed to answer the following two questions:

- *Effectiveness*: how effective are the proposed Opera algorithms at optimizing the connectivity measures (defined in the proposed SubLine family) of a multi-layered network (from the proposed MuLaN family)?
- *Efficiency*: how fast and scalable are our algorithms?

Experimental Setup

Data Sets Summary. We perform the evaluations on four different application domains, including (D1) a multi-layered Internet topology at the autonomous system level (MultiAS); (D2) critical infrastructure networks (InfraNet); (D3) a social-information collaboration network (SocInNet); and (D4) a biological CTD (Comparative Toxicogenomics Database) network Davis *et al.* (2015) (BIO). For the first two domains, we use real networks to construct the within-layer networks (i.e., $\mathcal{A}$ in the MuLaN model) and construct one or more cross-layer dependency structures based on real application scenarios (i.e., $\mathbf{G}$ and $\mathcal{D}$ in the MuLaN model). For the data sets in SocInNet and BIO domains, both the within-layer networks and cross-layer dependency networks are based on real connections. A summary of these data sets is shown in Table 5.4. We will present the detailed description of each application domains in Section 5.2.4.

Table 5.4: Data Sets Summary.

Data Sets	Application Domains	# of Layers	# of Nodes	# of Links
D1	MultiAS	2∼4	5,929∼24,539	11,183∼50,778
D2	InfraNet	3	19,235	46,926
D3	SocInNet	2	63,501∼124,445	13,097∼211,776
D4	BIO	3	35,631	253,827

Baseline Methods. To our best knowledge, there is no existing method which can be directly applied to the connectivity optimization problem (Problem 1) of the MuLaN model. We generate the baseline methods using two complementary strategies, including *forward propagation* ('FP' for short) and *backward propagation* ('BP' for short). The key idea behind the *forward propagation* strategy is that an important node in the *control layer* might have more impact on its dependent networks as well. On the other hand, for the *backward propagation strategy*, we first identify important nodes in the *target layer(s)*, and then trace back to its supporting layer(s) through the cross-layer dependency links (i.e., $\mathcal{D}$). For both strategies, we need a node importance measure. In our evaluations, we compare three different measures, including (1) node degree; (2) PageRank measure Page *et al.* (1998); and (3) *Netshield* values Tong *et al.* (2010). In addition, for comparison purpose, we also randomly select nodes either from the control layer (for the forward propagation strategy) or from the target layer(s) (for the backward propagation strategy). Altogether, we have eight baseline methods (four for each strategy, respectively), including (1) 'Degree-FP', (2) 'PageRank-FP', (3) 'Netshield-FP', (4) 'Rand-FP', (5) 'Degree-BP', (6) 'PageRank-

BP', (7) 'Netshield-BP', (8) 'Rand-BP'.

Opera Algorithms and Variants. We evaluate three prevalent network connectivity measures, including (1) the leading eigenvalue of the (within-layer) adjacency matrix, which relates to the epidemic threshold of a variety of cascading models; (2) the loop capacity (LC), which relates to the robustness of the network; and (3) the triangle capacity (TC), which relates to the local connectivity of the network. As mentioned in Chapter 3, both the loop capacity and the triangle capacity are members of the SubLine family. Strictly speaking, the leading eigenvalue does *not* belong to the SubLine family. Instead, it approximates the path capacity (PC), and the latter (PC) is a member of the SubLine family. Correspondingly, we have three instances of the proposed Opera algorithm (each corresponding to one specific connectivity measures) including 'Opera-PC', 'Opera-LC', and 'Opera-TC'. Recall that there is an optional lazy evaluation step (step 11) in the proposed Opera algorithm, thanks to the diminishing returns property of the SubLine connectivity measures. When the leading eigenvalue is chosen as the connectivity function, such diminishing returns property does not hold anymore. To address this issue, we introduce a variant of Opera-PC as follows. At each iteration, after the algorithm chooses a new node v (step 18, Algorithm 1), we (1) update the network by removing all the nodes that depend on node v, and (2) update the corresponding leading eigenvalues and eigenvectors. We refer to this variant as 'Opera-PC-Up'. For each of the three connectivity measures, we run all four Opera algorithms.

Machines and Repeatability. All the experiments are performed on a machine with 2 processors Intel Xeon 3.5GHz with 256GB of RAM. The algorithms are programmed with MATLAB using single thread. All the data sets used in this work are publicly available.

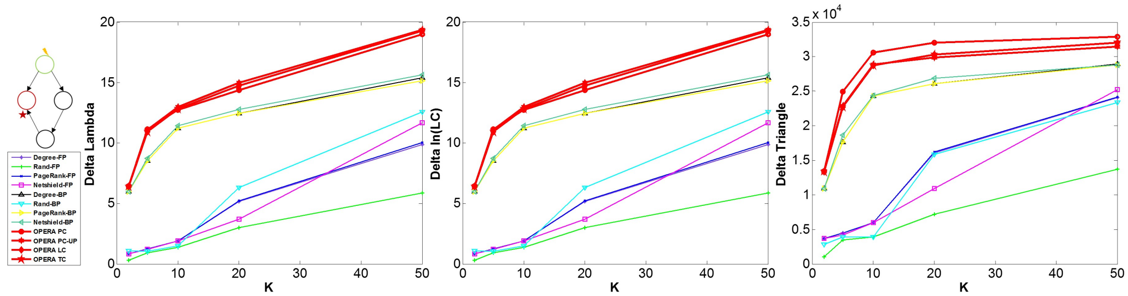


Figure 5.16: Evaluations on the MULTIAS Data Set, with a Four-layered Diamond-shaped Dependency Network. The Connectivity Change Vs. Budget. Larger Is Better. All the Four Instances of the Proposed OPERA Algorithm (in Red) Outperform the Baseline Methods.

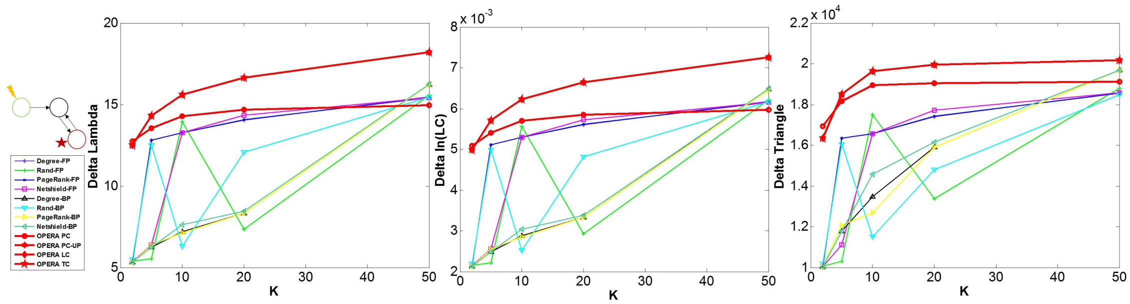


Figure 5.17: Evaluations on the MULTIAS Data Set, with a Three-layered Cyclic Dependency Network. The Connectivity Change Vs. Budget. Larger Is Better. Three out of Four Instances of the Proposed OPERA Algorithm (in Red) Outperform the Baseline Methods.

Effectiveness Results

D1 - MultiAS. This data set contains the Internet topology at the autonomous system level. The data set is available at `http://snap.stanford.edu/data/`. It has 9 different network snapshots, with $633 \sim 13,947$ nodes and $1,086 \sim 30,584$ edges. In our evaluations, we treat these snapshots as the within-layer adjacency matrices $\mathcal{A}$. For a given supporting layer, we generate the cross-layer node-node dependency matrices $\mathcal{D}$ by randomly choosing 3 nodes from its dependent layer as the direct dependents for each supporting node. For this application domain, we have experimented with different layer-layer dependency structures ($\mathbf{G}$), including a three-layered line-structured network, a three-layered tree-structured network, a four-layered diamond shaped network and a three-layered cyclic network. As the experimental results in the first three networks follow similar pattern, we only present the results on diamond-shaped network and cyclic network in Figure 5.16 and 5.17 due to page limits. Overall, the four instances of the proposed OPERA algorithm perform better than the baseline methods. Among the baseline methods, the *backward propagation* methods are better than the forward propagation methods under acyclic dependency networks (5.16). This is because the length of the backtracking path on the dependency network **G** (from the target layer to the control layer) is short. Therefore, compared with other baseline methods, the node set returned from the BP strategy is able to affect more important nodes in the target layer. While for the cyclic dependency network in Figure 5.17, the backtracking path is elongated by the cycle. Then the nodes selected by BP strategy are not guaranteed to affect more important nodes in the target layer than FP strategy.

D2 - InfraNet. This data set contains three types of critical infrastructure networks, including (1) the power grid, (2) the communication network; and (3) the airport networks. The power grid is an undirected, un-weighted network representing the topology of the Western States Power Grid of the United State Watts and Strogatz (1998). It has 4,941 nodes and 6,594 edges. We use one snapshot from the MULTIAS data set as the communication network with 11,461 nodes and 32,730 edges. The airport network represents the internal US air traffic lines between 2,649 airports and has 13,106 links (available at `http://www.levmuchnik.net/Content/Networks/NetworkData.html`). We construct a triangle-shaped layer-layer dependency network **G** (see the icon of Figure 5.18) based on the following observation. The operation of an airport depends on both the electricity provided by the power grid and the Internet support provided by the communication network. In the meanwhile, the full-functioning of the communication network depends on the support of power grid. We use similar strategy as in MultiAS to generate the cross-layer node-node dependency matrices $\mathcal{D}$. The results are summarized in Figure 5.18. Again, the proposed Opera algorithms outperform all the baseline methods. Similar to the MultiAS network, the backtracking path from the airport layer to the power grid layer is also very short. Therefore, the backward propagation strategies perform relatively better than other baseline methods. In addition, we change the density of the cross-layer node-node dependency matrices and evaluate its impact on the optimization results in Figure 5.19. We found that (1) across different dependency densities, the proposed Opera algorithms still outperform the baseline methods; and (2) when the dependency density increases, the algorithms lead to a larger decrease of the corresponding connectivity measures with the same budget.

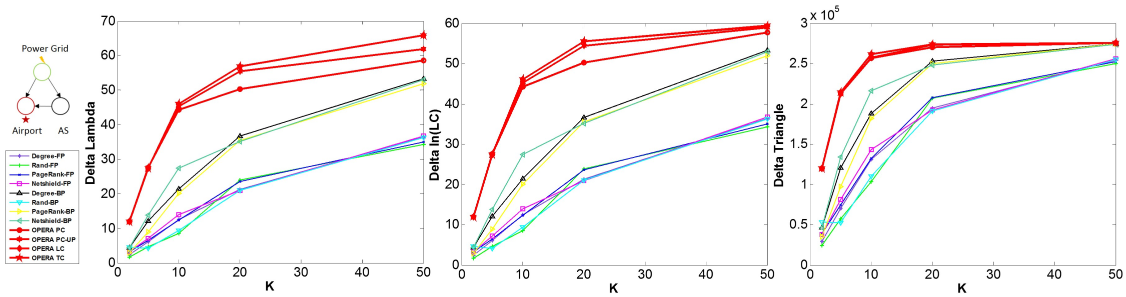


Figure 5.18: Evaluations on the InfraNet Data Set, with a Three-layered Triangle-shaped Dependency Network. The Connectivity Change Vs. Budget. Larger Is Better. All the Four Instances of the Proposed Opera Algorithm (in Red) Outperform the Baseline Methods.

D3 - SocInNet. This data set contains three types of social-information networks Tang *et al.* (2008), including (1) a co-authorship network; (2) a paper-paper citation network; and (3) a venue-venue citation network. Different from the previous two data sets, two types of cross-layer node-node dependency links naturally exist in this data set, including *who-writes-which paper*, and *which venue-publishes-which paper*. In our experiment, we use the papers published between year 1990 to

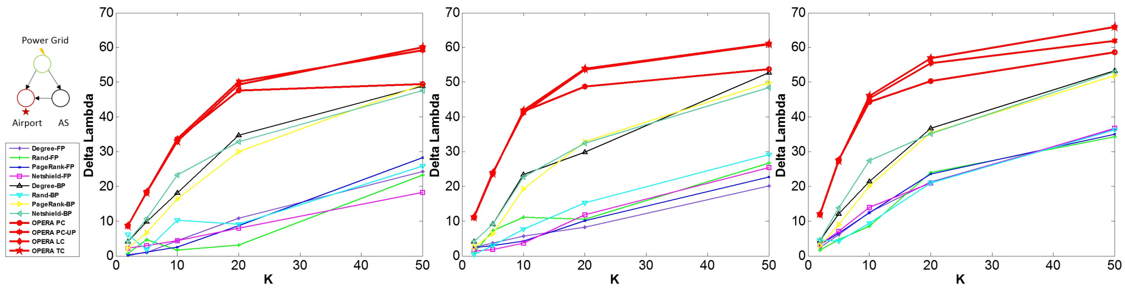


Figure 5.19: $\Delta\lambda$ W.R.T. k. Change the Average Number of Dependents Between Power Grid and as from 5, 10 to 15 (Left to Right).

1992. In total, there are 62,602 papers, 61,843 authors, 899 venues, 10,739 citation links, 201,037 collaboration links, 2,358 venue citation links, 126,242 author-paper cross-layer links, and 62,602 venue-paper cross-layer links.

We evaluate the proposed algorithms in two scenarios with this data set, including (1) an author-paper two-layered network; and (2) a venue-paper two-layered network. For both scenarios, we choose the paper-paper citation network as the target layer. Figure 5.20 presents the results on the author-paper two-layered network. We can see that three out of four Opera algorithms outperform all the baseline methods in all the three cases. Opera-PC does not perform as well as the remaining three Opera instances due to the gap between the leading eigenvalue and the actual path capacity. However, the issue can be partially addressed with Opera-PC-Up by introducing an update step. Among the baseline methods, the *backward propagation* strategy is better since the target layer is directly dependent on the control layer, which makes it possible to trace back the high-impact authors given the set of high-impact papers. The poor performance of the *forward propagation* methods implies that a socially active author does not necessarily have high-impact papers. The results on the venue-paper network is similar as shown in Figure 5.21. Different from the author-paper network, the backward propagation strategies perform worse than the forward propagation strategies. This is probably due to the fact that not all the important (high-impact) papers appear in the important (high-impact) venues.

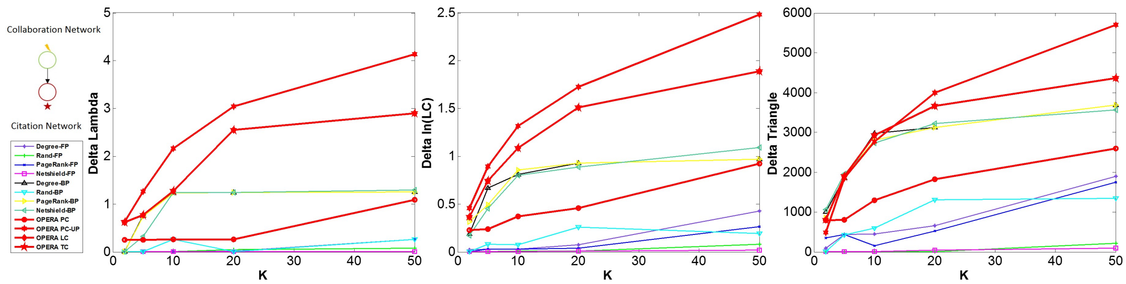


Figure 5.20: Evaluations on the SocInNet Data Set, with a Two-layered Author-paper Dependency Network. The Connectivity Change Vs. Budget. Larger Is Better. Three out of Four Proposed Opera Algorithms (in Red) Outperform the Baseline Methods.

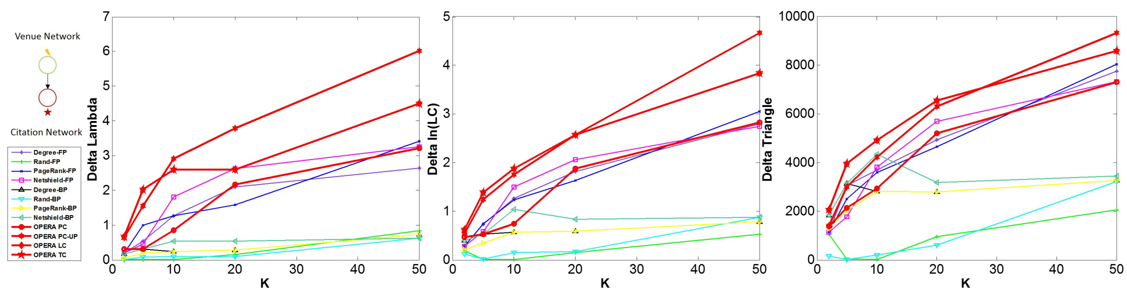


Figure 5.21: Evaluations on the SocInNet Data Set, with a Two-layered Venue-paper Dependency Network. The Connectivity Change Vs. Budget. Larger Is Better. Three out of Four Proposed Opera Algorithms (in Red) Outperform the Baseline Methods.

D4 - BIO. This data set contains three types of biological networks Davis *et al.* (2015) including (1) a chemical similarity network with 6,026 chemicals, 69,109 links; (2) a gene similarity network with 25,394 genes, 154,167 links; and (3) a disease similarity network with 4,256 diseases, 30,551 links. The dependencies between those layers depict *which chemical-affects-which gene*, *which chemical-treats-which disease*, and *which gene-associates-which disease* relations, each of which contains 53,735, 19,771 and 1,950 dependency links respectively. The evaluation results are as shown in Figure 5.22. Despite the fact that the proposed Opera algorithms outperform all other baseline methods, there are two interesting observations that worth to be mentioned. First is that the impact of chemical nodes on disease networks become saturated at a small budge value for all connectivity measures, which implies that only a few chemicals are effective in treating most of the diseases in the given data set. Second, the ineffectiveness of *forward propagation* methods indicates that chemicals with various compounds (high within-layer centrality nodes) may have little effects in disease treatment.

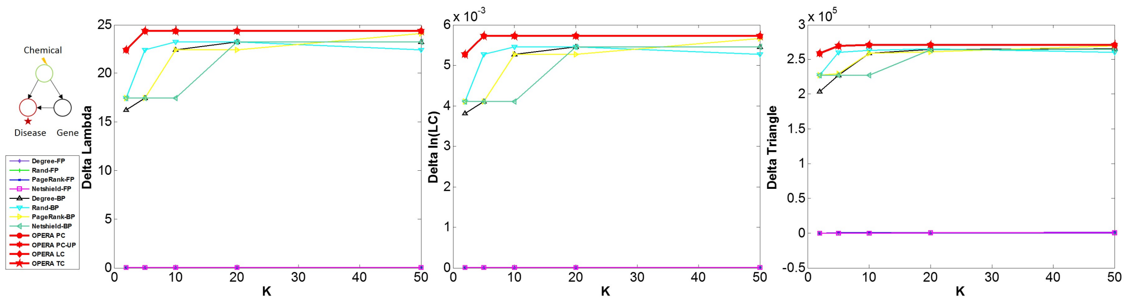


Figure 5.22: Evaluations on the BIO Data Set, with a Three-layered Triangle-shaped Dependency Network. The Connectivity Change Vs. Budget. Larger Is Better. All Four Proposed OPERA Algorithms (in Red) Outperform the Baseline Methods.

Efficiency Results

Figure 5.23 presents the scalability of the proposed OPERA algorithms. We can see that all four instances of OPERA scale linearly with respect to the size of the input multi-layered network (i.e., $N + M + L$), which is consistent with our complexity analysis. The wall-clock time for OPERA-PC-Up is the longest compared with the remaining three instances, due to the additional update step.

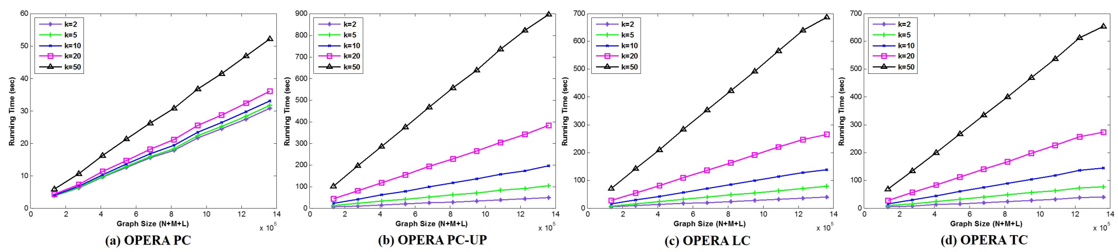


Figure 5.23: Wall-clock Time Vs. The Size of the Input Networks. The Proposed OPERA Algorithms Scale Linearly W.R.T. $(N + M + L)$.

Chapter 6

CONCLUSION AND FUTURE WORK

In this chapter, we summarize our key research results and discuss future research directions for network connectivity.

6.1 Conclusion

In this dissertation, we propose three main tasks for network connectivity studies, which includes (1) connectivity measures, (2) connectivity inference and (3) connectivity optimization.

Measures. For connectivity measures, our main finding is that various task-oriented connectivity measures in the literature can be unified into a generalized model — the SubLine connectivity model. The key idea of SubLine model is to view the connectivity of the network as the aggregation of the connectivity of some valid subgraphs. By restricting the valid subgraphs to a subset of nodes, the SubLine connectivity can be used to measure the local connectivity of a subnetwork. Moreover, we also show that the proposed model can be easily extended to multi-layered networks.

Inference. For connectivity inference, we addressed the eigen-functions/connectivity tracking problem in dynamic networks and the cross-layer dependency inference problem in multi-layered networks. To efficiently track the eigen-functions in the network, we propose Trip-Basic and Trip. In addition, we provide a framework for attribution analysis on eigen-functions and a method to effectively estimate tracking errors. Our experiments show that both Trip-Basic and Trip can effectively and efficiently track the changes of eigen-pairs, number of triangles, robustness score and eigen-gap in dynamic graphs, while Trip is more stable over time. In both cases, the accumulated error rate inevitably keeps increasing as time goes by. As for the dependency inference problem, we propose to formulate the inference problem as a collective collaborative filtering problem and introduce FASCINATE, an algorithm that can effectively infer the missing dependencies with provable optimality and scalability. In particular, by modeling the impact of zero-start node as a perturbation in the multi-layered network, we derive FASCINATE-ZERO, an online variant of FASCINATE that can approximate the dependencies of the newly added node with sub-linear complexity w.r.t. the overall system size. The experimental results on five real-world datasets demonstrate the superiority of our proposed algorithm both by its effectiveness and efficiency.

Optimization. For the connectivity optimization task, we first prove that for any network connectivity measures in the SUBLINE family, the connectivity optimization problem with the MULAN model enjoys the diminishing returns property, which naturally lends itself to a family of provable near-optimal algorithms using greedy scheme. Then we show that a wide range of network connectivity optimization (NETCOM) problems are NP-complete and $(1 - 1/e)$ is the best approximation ratio that a polynomial algorithm can achieve for NETCOM problems unless $NP \subseteq DTIME(n^{O(\log\log n)})$. On the algorithmic aspect, we propose a series of effective, scalable and generalizable optimization algorithms CONTAIN and OPERA that can be applied to both single-layered networks and multi-layered networks.

6.2 Future Work

Network connectivity is a powerful graph parameter which may lead to many interesting findings. Below we present some promising research directions:

- **Multi-layered Network Connectivity.** In our work, we model the connectivity of a multi-layered network as an aggregation over the connectivity on each layers. This method omits the dependency structure across the domains, which includes rich information to be exploited. To start with, the dependency between nodes may bear different types. For example, the functioning of a node may require the functioning of all its supporting node (the dependency type used in our work), or the functioning of only one of its supporting node. The robustness scores of two networks with the same structure but different dependency types would vary a lot from each other. Thus, how to effectively model the connectivity in such more informed multi-layered networks is an interesting direction to discover.

- **Dynamic Network Inference.** Real-world networks are evolving over time. In the infrastructure system, new power stations are constantly being added to the system to fulfill the increasing need for electricity supply. Similar expanding process can be observed in autonomous systems and transportation networks as well. Simultaneously, new cross-layer dependencies across those layers would be established in the system as well. The changing structure would inevitably affect the inference results from the old system. Therefore, it is necessary to re-calibrate the inference results timely to accommodate system changes. The naive way to solve the dynamic network inference problem is to re-run the static inference algorithms whenever the system changes. However, such strategy would be very inefficient when the system is changing fast. Although numerous dynamic algorithms have been proposed single-layered networks, few attempts have been made on multi-layered networks.

- **Connectivity Optimization and Adversarial Attack.** The adversarial attack on networked data has become a trending topic in recent years. Its main idea is to alter the network structure and node attributes to affect the results of subsequent tasks to the maximum extent. We speculate that the connectivity optimization algorithms may be helpful on some of the adversarial attack tasks as the nodes/edges selected by our algorithm are the ones that have largest impact on the overall network connectivity, which is an important parameter in many graph mining tasks.

REFERENCES

Albert, R., H. Jeong and A.-L. Barabási, "Error and attack tolerance of complex networks", Nature **406**, 6794, 378–382 (2000).

Battiston, F., V. Nicosia and V. Latora, "Structural measures for multiplex networks", Physical Review E **89**, 3, 032804 (2014).

Benson, A. R., R. Abebe, M. T. Schaub, A. Jadbabaie and J. Kleinberg, "Simplicial closure and higher-order link prediction", arXiv preprint arXiv:1802.06916 (2018).

Berlingerio, M., M. Coscia, F. Giannotti, A. Monreale and D. Pedreschi, "Foundations of multidimensional network analysis", in "Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on", pp. 485–489 (IEEE, 2011).

Boccaletti, S., G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang and M. Zanin, "The structure and dynamics of multilayer networks", Physics Reports **544**, 1, 1–122 (2014).

Buldyrev, S. V., R. Parshani, G. Paul, H. E. Stanley and S. Havlin, "Catastrophic cascade of failures in interdependent networks", Nature **464**, 7291, 1025–1028 (2010).

Chakrabarti, D., Y. Wang, C. Wang, J. Leskovec and C. Faloutsos, "Epidemic thresholds in real networks", ACM Transactions on Information and System Security (TISSEC) **10**, 4, 1 (2008).

Chan, H., L. Akoglu and H. Tong, "Make it or break it: manipulating robustness in large networks", in "Proceedings of 2014 SIAM International Conference on Data Mining", pp. 325–333 (SIAM, 2014).

Chen, C., J. He, N. Bliss and H. Tong, "On the connectivity of multi-layered networks: Models, measures and optimal control", in "Data Mining (ICDM), 2015 IEEE 15th International Conference on", pp. 715–720 (IEEE, 2015).

Chen, C., R. Peng, L. Ying and H. Tong, "Network connectivity optimization: Fundamental limits and effective algorithms", in "Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining", pp. 1167–1176 (ACM, 2018).

Chen, C. and H. Tong, "On the eigen-functions of dynamic graphs: Fast tracking and attribution algorithms", Statistical Analysis and Data Mining: The ASA Data Science Journal **10**, 2, 121–135 (2017).

Chen, C., H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos and C. Faloutsos, "Eigen-optimization on large graphs by edge manipulation", TKDD **10**, 4, 49, URL `http://doi.acm.org/10.1145/2903148` (2016a).

Chen, C., H. Tong, B. A. Prakash, C. E. Tsourakakis, T. Eliassi-Rad, C. Faloutsos and D. H. Chau, "Node immunization on large graphs: Theory and algorithms", IEEE Transactions on Knowledge and Data Engineering **28**, 1, 113–126 (2016b).

Chen, C., H. Tong, L. Xie, L. Ying and Q. He, "FASCINATE: fast cross-layer dependency inference on multi-layered networks", in "Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016", pp. 765–774 (2016c), URL `http://doi.acm.org/10.1145/2939672.2939784`.

Chen, L., X. Xu, S. Lee, S. Duan, A. G. Tarditi, S. Chinthavali and B. A. Prakash, "Hotspots: Failure cascades on heterogeneous critical infrastructure networks", in "Proceedings of the 2017 ACM on Conference on Information and Knowledge Management", pp. 1599–1607 (ACM, 2017).

Chen, P.-Y. and A. O. Hero, "Local fiedler vector centrality for detection of deep and overlapping communities in networks", in "Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on", pp. 1120–1124 (IEEE, 2014).

Chen, W., W. Hsu and M. L. Lee, "Making recommendations from multiple domains", in "Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 892–900 (ACM, 2013).

Chen, W., C. Wang and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks", in "Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 1029–1038 (ACM, 2010).

Chen, W., Y. Wang and S. Yang, "Efficient influence maximization in social networks", in "Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 199–208 (ACM, 2009).

Chung, F. R., *Spectral graph theory*, vol. 92 (American Mathematical Soc., 1997).

Cohen, R., S. Havlin and D. Ben-Avraham, "Efficient immunization strategies for computer networks and populations", Physical review letters **91**, 24, 247901 (2003).

Davis, A. P., C. J. Grondin, K. Lennon-Hopkins, C. Saraceni-Richards, D. Sciaky, B. L. King, T. C. Wiegers and C. J. Mattingly, "The comparative toxicogenomics database's 10th year anniversary: update 2015", Nucleic acids research **43**, D1, D914–D920 (2015).

De Domenico, M., A. Solé-Ribalta, E. Cozzo, M. Kivelä, Y. Moreno, M. A. Porter, S. Gómez and A. Arenas, "Mathematical formulation of multilayer networks", Physical Review X **3**, 4, 041022 (2013).

De Domenico, M., A. Solé-Ribalta, E. Omodei, S. Gómez and A. Arenas, "Ranking in interconnected multilayer networks reveals versatile nodes", Nature communications **6** (2015).

Ding, C., T. Li, W. Peng and H. Park, "Orthogonal nonnegative matrix t-factorizations for clustering", in "Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 126–135 (ACM, 2006).

Drineas, P. and M. W. Mahoney, "On the nyström method for approximating a gram matrix for improved kernel-based learning", The Journal of Machine Learning Research **6**, 2153–2175 (2005).

Faloutsos, M., P. Faloutsos and C. Faloutsos, "On power-law relationships of the internet topology", in "ACM SIGCOMM computer communication review", vol. 29, pp. 251–262 (ACM, 1999).

Frank, H. and I. Frisch, "Analysis and Design of Survivable Networks", Communication Technology, IEEE Transactions on **18**, 5, 501–519 (1970).

Freeman, L. C., "A set of measures of centrality based on betweenness", Sociometry pp. 35–41 (1977).

Freeman, L. C., "Centrality in social networks conceptual clarification", Social networks **1**, 3, 215–239 (1978).

Gao, J., S. V. Buldyrev, S. Havlin and H. E. Stanley, "Robustness of a network of networks", Physical Review Letters **107**, 19, 195701 (2011).

Gao, J., S. V. Buldyrev, H. E. Stanley and S. Havlin, "Networks formed from interdependent networks", Nature physics **8**, 1, 40–48 (2012).

Harary, F. and A. Schwenk, "The spectral approach to determining the number of walks in a graph", Pacific Journal of Mathematics **80**, 2, 443–449 (1979).

Heath, L. S. and A. A. Sioson, "Multimodal networks: Structure and operations", Computational Biology and Bioinformatics, IEEE/ACM Transactions on **6**, 2, 321–332 (2009).

Hoory, S., N. Linial and A. Wigderson, "Expander graphs and their applications", Bulletin of the American Mathematical Society **43**, 4, 439–561 (2006).

Hu, Y., Y. Koren and C. Volinsky, "Collaborative filtering for implicit feedback datasets", in "Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on", pp. 263–272 (Ieee, 2008).

Jerrum, M. and A. Sinclair, "Conductance and the rapid mixing property for markov chains: the approximation of permanent resolved", in "Proceedings of the twentieth annual ACM symposium on Theory of computing", pp. 235–244 (ACM, 1988).

Jun, W., M. Barahona, T. Yue-Jin and D. Hong-Zhong, "Natural connectivity of complex networks", Chinese Physics Letters **27**, 7, 078902 (2010).

Kempe, D., J. Kleinberg and É. Tardos, “Maximizing the spread of influence through a social network”, in “Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining”, pp. 137–146 (ACM, 2003).

Khuller, S., A. Moss and J. S. Naor, “The budgeted maximum coverage problem”, Information Processing Letters **70**, 1, 39–45 (1999).

Kivelä, M., A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno and M. A. Porter, “Multilayer networks”, Journal of Complex Networks **2**, 3, 203–271 (2014).

Kleinberg, J. and E. Tardos, *Algorithm design* (Pearson Education India, 2006).

Kleinberg, J. M., “Authoritative sources in a hyperlinked environment”, in “ACM-SIAM Symposium on Discrete Algorithms”, (1998).

Kohanski, M. A., D. J. Dwyer and J. J. Collins, “How antibiotics kill bacteria: from targets to networks”, Nature Reviews Microbiology **8**, 6, 423 (2010).

Koren, Y., R. Bell and C. Volinsky, “Matrix factorization techniques for recommender systems”, Computer , 8, 30–37 (2009).

Kovacs, I. A. and A.-L. Barabasi, “Network science: Destruction perfected”, Nature **524**, 7563, 38–39 (2015).

Le, L. T., T. Eliassi-Rad and H. Tong, “Met: A fast algorithm for minimizing propagation in large graphs with small eigen-gaps”, in “Proceedings of the 2015 SIAM International Conference on Data Mining”, pp. 694–702 (SIAM, 2015).

Lee, D. D. and H. S. Seung, “Algorithms for non-negative matrix factorization”, in “Advances in neural information processing systems”, pp. 556–562 (2001).

Leskovec, J., L. A. Adamic and B. A. Huberman, “The dynamics of viral marketing”, ACM Transactions on the Web (TWEB) **1**, 1, 5 (2007a).

Leskovec, J., J. Kleinberg and C. Faloutsos, “Graphs over time: densification laws, shrinking diameters and possible explanations”, in “Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining”, pp. 177–187 (ACM, 2005).

Leskovec, J., J. Kleinberg and C. Faloutsos, “Graph evolution: Densification and shrinking diameters”, ACM Transactions on Knowledge Discovery from Data (TKDD) **1**, 1, 2 (2007b).

Leskovec, J., A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen and N. Glance, “Cost-effective outbreak detection in networks”, in “Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining”, pp. 420–429 (ACM, 2007c).

Li, B., Q. Yang and X. Xue, “Can movies and books collaborate? cross-domain collaborative filtering for sparsity reduction.”, in “IJCAI”, vol. 9, pp. 2052–2057 (2009).

Li, J., X. Hu, L. Wu and H. Liu, "Robust unsupervised feature selection on networked data", in "Proceedings of the 2016 SIAM International Conference on Data Mining", pp. 387–395 (SIAM, 2016).

Li, L., H. Tong, N. Cao, K. Ehrlich, Y.-R. Lin and N. Buchler, "Replacing the irreplaceable: Fast algorithms for team member recommendation", in "Proceedings of the 24th International Conference on World Wide Web", pp. 636–646 (ACM, 2015a).

Li, L., H. Tong, Y. Xiao and W. Fan, "Cheetah: Fast graph kernel tracking on dynamic graphs", in "SDM", (SIAM, 2015b).

Li, R.-H. and J. X. Yu, "Triangle minimization in large networks", Knowledge and Information Systems **45**, 3, 617–643 (2015).

Li, Y., J. Hu, C. Zhai and Y. Chen, "Improving one-class collaborative filtering by incorporating rich user information", in "Proceedings of the 19th ACM international conference on Information and knowledge management", pp. 959–968 (ACM, 2010).

Lin, C.-b., "Projected gradient methods for nonnegative matrix factorization", Neural computation **19**, 10, 2756–2779 (2007).

Liu, J., C. Wang, J. Gao, Q. Gu, C. C. Aggarwal, L. M. Kaplan and J. Han, "Gin: A clustering model for capturing dual heterogeneity in networked data", in "SDM", pp. 388–396 (SIAM, 2015).

Lu, Z., W. Pan, E. W. Xiang, Q. Yang, L. Zhao and E. Zhong, "Selective transfer learning for cross domain recommendation", in "SDM", pp. 641–649 (SIAM, 2013).

Ma, H., D. Zhou, C. Liu, M. R. Lyu and I. King, "Recommender systems with social regularization", in "Proceedings of the 4th ACM International Conference on Web Search and Data Mining", pp. 287–296 (2011).

Mcauley, J. and J. Leskovec, "Discovering social circles in ego networks", ACM Transactions on Knowledge Discovery from Data (TKDD) **8**, 1, 4 (2014).

Milo, R., S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii and U. Alon, "Network motifs: simple building blocks of complex networks", Science **298**, 5594, 824–827 (2002).

Moody, J. and D. R. White, "Social cohesion and embeddedness: A hierarchical conception of social groups", American Sociological Review pp. 1–25 (2003).

Morone, F. and H. A. Makse, "Influence maximization in complex networks through optimal percolation", Nature **524**, 7563, 65 (2015).

Nemhauser, G. L., L. A. Wolsey and M. L. Fisher, "An analysis of approximations for maximizing submodular set functionsi", Mathematical Programming **14**, 1, 265–294 (1978).

Newman, M. E., “A measure of betweenness centrality based on random walks”, Social networks **27**, 1, 39–54 (2005).

Newman, M. E., “The mathematics of networks”, The new palgrave encyclopedia of economics **2**, 2008, 1–12 (2008).

Ni, J., H. Tong, W. Fan and X. Zhang, “Inside the atoms: ranking on a network of networks”, in “Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining”, pp. 1356–1365 (ACM, 2014).

Ning, H., W. Xu, Y. Chi, Y. Gong and T. S. Huang, “Incremental spectral clustering by efficiently updating the eigen-system”, Pattern Recognition **43**, 1, 113–127 (2010).

Page, L., S. Brin, R. Motwani and T. Winograd, “The PageRank citation ranking: Bringing order to the web”, Tech. rep., Stanford Digital Library Technologies Project, URL `http://dbpubs.stanford.edu/pub/1999-66`, paper SIDL-WP-1999-0120 (version of 11/11/1999) (1998).

Pan, R., Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz and Q. Yang, “One-class collaborative filtering”, in “Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on”, pp. 502–511 (IEEE, 2008).

Parshani, R., S. V. Buldyrev and S. Havlin, “Interdependent networks: reducing the coupling strength leads to a change from a first to second order percolation transition”, Physical review letters **105**, 4, 048701 (2010).

Prakash, B. A., D. Chakrabarti, N. C. Valler, M. Faloutsos and C. Faloutsos, “Threshold conditions for arbitrary cascade models on arbitrary networks”, Knowledge and information systems **33**, 3, 549–575 (2012).

Prakash, B. A., A. Sridharan, M. Seshadri, S. Machiraju and C. Faloutsos, “Eigenspokes: Surprising patterns and scalable community chipping in large graphs”, in “Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II”, pp. 435–448 (2010).

Razick, S., G. Magklaras and I. M. Donaldson, “irefindex: a consolidated protein interaction database with provenance”, BMC bioinformatics **9**, 1, 1 (2008).

Rinaldi, S. M., J. P. Peerenboom and T. K. Kelly, “Identifying, understanding, and analyzing critical infrastructure interdependencies”, Control Systems, IEEE **21**, 6, 11–25 (2001).

Rosato, V., L. Issacharoff, F. Tiriticco, S. Meloni, S. Porcellinis and R. Setola, “Modelling interdependent infrastructures using interacting dynamical models”, International Journal of Critical Infrastructures **4**, 1-2, 63–79 (2008).

Sánchez-García, R. J., E. Cozzo and Y. Moreno, “Dimensionality reduction and spectral properties of multilayer networks”, Physical Review E **89**, 5, 052815 (2014).

Sen, A., A. Mazumder, J. Banerjee, A. Das and R. Compton, “Multi-layered network using a new model of interdependency”, arXiv preprint arXiv:1401.1783 (2014).

Shao, J., S. V. Buldyrev, S. Havlin and H. E. Stanley, “Cascade of failures in coupled network systems with multiple support-dependence relations”, Physical Review E **83**, 3, 036116 (2011).

Singh, A. P. and G. J. Gordon, “Relational learning via collective matrix factorization”, in “Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining”, pp. 650–658 (ACM, 2008).

Sipser, M., *Introduction to the theory of computation* (PWS Publishing Company, 1997).

Stewart, G. W. and J.-G. Sun, *Matrix Perturbation Theory* (Academic Press, 1990).

Tang, J., H. Gao and H. Liu, “mtrust: discerning multi-faceted trust in a connected world”, in “Proceedings of the 5th ACM International Conference on Web Search and Data Mining”, pp. 93–102 (2012a).

Tang, J., H. Gao, H. Liu and A. Das Sarma, “etrust: Understanding trust evolution in an online world”, in “Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining”, pp. 253–261 (2012b).

Tang, J., J. Zhang, L. Yao, J. Li, L. Zhang and Z. Su, “Arnetminer: extraction and mining of academic social networks”, in “Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining”, pp. 990–998 (ACM, 2008).

Tarjan, R., “Depth-first search and linear graph algorithms”, SIAM journal on computing **1**, 2, 146–160 (1972).

Tong, H., S. Papadimitriou, P. S. Yu and C. Faloutsos, “Fast monitoring proximity and centrality on time-evolving bipartite graphs”, Statistical Analysis and Data Mining **1**, 3, 142–156 (2008).

Tong, H., B. A. Prakash, T. Eliassi-Rad, M. Faloutsos and C. Faloutsos, “Gelling, and melting, large graphs by edge manipulation”, in “Proceedings of the 21st ACM international conference on Information and knowledge management”, pp. 245–254 (ACM, 2012).

Tong, H., B. A. Prakash, C. Tsourakakis, T. Eliassi-Rad, C. Faloutsos and D. H. Chau, “On the vulnerability of large graphs”, in “Data Mining (ICDM), 2010 IEEE 10th International Conference on”, pp. 1091–1096 (IEEE, 2010).

Tsourakakis, C. E., “Fast counting of triangles in large real networks without counting: Algorithms and laws”, in “Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on”, pp. 608–617 (IEEE, 2008).

Van Driel, M. A., J. Bruggeman, G. Vriend, H. G. Brunner and J. A. Leunissen, "A text-mining analysis of the human phenome", European journal of human genetics **14**, 5, 535–542 (2006).

Vespignani, A., "Complex networks: The fragility of interdependency", Nature **464**, 7291, 984–985 (2010).

Wang, Y., D. Chakrabarti, C. Wang and C. Faloutsos, "Epidemic spreading in real networks: An eigenvalue viewpoint", in "Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on", pp. 25–34 (IEEE, 2003).

Wasserman, S., *Social network analysis: Methods and applications*, vol. 8 (Cambridge university press, 1994).

Watts, D. J. and S. H. Strogatz, "Collective dynamics of small-world networks", nature **393**, 6684, 440–442 (1998).

Williams, V. V., "Breaking the coppersmith-winograd barrier", (2011).

Wu, J., B. Mauricio, Y.-J. Tan and H.-Z. Deng, "Natural connectivity of complex networks", Chinese Physics Letters **27**, 7, 78902 (2010).

Xu, C., D. Tao and C. Xu, "A survey on multi-view learning", arXiv preprint arXiv:1304.5634 (2013).

Yang, D., J. He, H. Qin, Y. Xiao and W. Wang, "A graph-based recommendation across heterogeneous domains", in "Proceedings of the 24rd ACM International Conference on Conference on Information and Knowledge Management", pp. 463–472 (ACM, 2015).

Yao, Y., H. Tong, G. Yan, F. Xu, X. Zhang, B. K. Szymanski and J. Lu, "Dual-regularized one-class collaborative filtering", in "Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management", pp. 759–768 (ACM, 2014).

Yin, H., A. R. Benson and J. Leskovec, "The local closure coefficient: A new perspective on network clustering", networks **26**, 41, 44 (2019).

Yin, H., A. R. Benson, J. Leskovec and D. F. Gleich, "Local higher-order graph clustering", in "Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", pp. 555–564 (2017), URL `http://doi.acm.org/10.1145/3097983.3098069`.

Zhou, D., J. He, K. S. Candan and H. Davulcu, "MUVIR: multi-view rare category detection", in "Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015", pp. 4098–4104 (2015), URL `http://ijcai.org/Abstract/15/575`.