

© 2022 Boxin Du

MULTI-NETWORK ASSOCIATION: ALGORITHMS AND APPLICATIONS

BY

BOXIN DU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Doctoral Committee:

Associate Professor Hanghang Tong, Chair
Professor Arindam Banerjee
Professor Chengxiang Zhai
Professor Jian Tang, Mila-Quebec AI Institute and HEC Montreal

ABSTRACT

Networks extracted from multiple sources and platforms or from multiple instances of identical domains form the multi-network, such as large social networks collected from Facebook and Instagram, medium-scaled networks of chemical compound and proteins extracted from chemical/protein interaction, etc. Multi-network association refers to the node associations or proximities in a multi-network model, which goes beyond the boundary of node associations of a single simple network. Multi-network association offers a fundamental primitive for mining multi-networks, in the sense that it reveals the unique, collective relations among node sets, which can not be captured by mining individual networks separately.

Although network mining has become a ubiquitous tool of knowledge discovery for researchers and practitioners in diverse application domains, research in the multi-network association is still relatively limited, owing to the following three major challenges. First (*Problem formulation*), how do we explicitly formulate the multi-network association inference problem in various multi-network scenarios, such as multiple plain/attributed networks, multi-layered networks, hypergraphs, etc.? How do we implicitly preserve multi-network association in an embedding model which is targeted at multi-network mining tasks? Second (*Computational complexity*), how can we develop efficient algorithms for mitigating the high complexity of the problems defined on multi-networks, in terms of both time and space complexity? Third (*Application*), how will the multi-network association empower or enable novel applications on multi-network data? To what extent can the multi-network association based methods boost the classic multi-network mining tasks?

In this Ph.D. thesis, an in-depth study of the multi-network association is formally discussed and analyzed to jointly tackle the aforementioned challenges. Specially, the research works from this thesis are organized based on the taxonomy of the network associations (i.e. pairwise vs. high-order association), and the taxonomy of the core algorithmic techniques (i.e. numerical vs. neural methods). First (*pairwise association with numerical techniques*), we develop a family of fast solvers (FASTEN) for the Sylvester equation, which lays the foundation of numerous multi-network mining tasks. We further introduce a novel application, namely interactive subgraph matching, empowered by the Sylvester equation. Second (*pairwise association with neural techniques*), we extend the boundary of the numerical techniques for pairwise association and design Sylvester Multi-Graph Neural Network model. As a generalization of traditional Sylvester equation, its flexible architecture could incorporate numerical features, and it is able to be adapted to various downstream tasks. We then show

that how such technique could be successfully applied on the application of social recommendation, to achieve up to 30% improvement over baseline methods. Third (*high-order association with numerical techniques*), we design a family of algorithms (i.e., SYTE) for multi-way association problem on both plain and attributed networks. It shows applicability in a variety of multi-network mining tasks, such as multi-network alignment. Forth (*high-order association with neural techniques*), we develop an unsupervised multi-resolution multi-network embedding model to simultaneously embed network elements of different resolutions and different networks into the same embedding space. We also present a hypergraph representation learning model via pre-training strategy, with a real-world case study on the inconsistent variation family detection problem for Amazon selection and catalog system.

This thesis is dedicated to my parents, Wenhong Zheng and Rongsan Du.

ACKNOWLEDGMENTS

My five-year Ph.D. journey is a winding, bumpy, yet fruitful one for me. First of all, I feel extremely grateful and lucky that I could conduct all my research under the guidance of my advisor Dr. Hanghang Tong. I was a slow student in the beginning of my Ph.D. study, but Dr. Tong advised me with great patience and high standard. During my first paper submission, I was astonished by the extent of his focus on details and his rigorous standard and attitude on research, because he could always instantly and accurately point out issues in my paper. I had to work so hard to reach his requirements. The paper was finally accepted by ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) in 2017 with unanimous high scores in the review, but what I learned from it went far beyond how to write a paper. The critical thinking, the ability to absorb substantial knowledge in a short period to locate key problems, and the high academic standard laid the solid foundation of my later research. During my Ph.D. study, Besides research, Dr. Tong provided me with many great opportunities in my academic training, such as our four-year Modeling Adversarial Activity (MAA) projects, teaching and mentoring experiences, writing proposals, and attending PI meetings and academic conferences. For instance, in teaching, I had the chance from giving lectures on graduate courses to mentoring students on research. I found myself applying some tips of his mentoring method into my own mentorship experience. Thanks to Dr. Tong's careful guidance, I was equipped with adequate research skills and mental confidence, which were so crucial for me to go through this long journey. I am sure that I will definitely continue to benefit from his wisdom in my future career, and I will be continuously thankful for what I have learned from him.

I would like to thank the rest of my committee members, Dr. Arindam Banerjee, Dr. Jian Tang, and Dr. Chengxiang Zhai for providing me with insightful feedback and suggestions during my thesis proposal and defense. Specifically, I would like to thank Dr. Chengxiang Zhai for sharing his broad knowledge and valuable opinions in the CS 510 course (Advanced Information Retrieval), which I took in Fall 2019. I would like to thank Dr. Arindam Banerjee for his feedback and pointing me to related works of applying the Sylvester equation in data mining, which broadens my horizon. I would like to thank Dr. Jian Tang for his pioneering works in network embedding. Inspired by his pilot works, I became highly interested in the network embedding field and conducted my own research in this direction as well.

I am also quite lucky to be able to finish two research internships during my Ph.D. study.

I worked with WeWork in Summer 2019 and with Amazon in Summer 2020, respectively. I would like to thank my mentor and my manager: Shengqi Yang from WeWork internship, and Changhe Yuan and B Narendran from Amazon internship. Also, I would like to thank all my collaborators from Amazon Relation Science Team: Robert Barton and Tal Neiman. My work of internship was published in Amazon's internal machine learning conference in 2021, and made great impact therein.

It is really fun and inspiring to work with all the labmates from the two labs led by Dr. Hanghang Tong and Dr. Jingrui He. Our discussions, collaborations, mutual support in and out of the research life are all invaluable treasure, and I am thankful for that. Our lab members include: Liangyue Li, Chen Chen, Si Zhang, Qinghai Zhou, Jian Kang, Zhe Xu, Lihui Liu, Yuchen Yan, Baoyu Jing, Yuheng Zhang, Shengyu Feng, Derek Wang, Yian Wang, Zhichen Zeng, Zongyu Lin, Shweta Jain, Scott Freitas, Haichao Yu, Ruiyue Peng, Rongyu Lin, Xiaoyu Zhang, Dawei Zhou, Yao Zhou, Arun Reddy, Xu Liu, Jun Wu, Lecheng Zheng, Xue Hu, Pei Yang, Dongqi Fu, Yikun Ban, Yunzhe Qi, Haonan Wang, Ziwei Wu, Wenxuan Bao, Tianxin Wei. I am really happy for Dr. Tong and Dr. He that this roster has grown so long since I joined as one of the early members, and I have no doubt that it will become longer and longer in the future. I would like to thank all my friends from Arizona State University and University of Illinois Urbana-Champaign.

Last but not least, I would like to thank my parents a million, for the unconditional love, support, and far-reaching vision. I would also like to thank my girlfriend Meiming Quan for her affectionate love, steadfast support and company.

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	1
1.1	Motivation	1
1.2	Research Challenges	2
1.3	Research Task Overview	3
1.4	Thesis Organization	6
CHAPTER 2	LITERATURE REIVEW	8
2.1	Pairwise Network Association	8
2.2	High-Order Network Association	9
2.3	Multi-Network Association Applications	12
2.4	Limitations for Existing Works	15
CHAPTER 3	PAIRWISE ASSOCIATION WITH NUMERICAL TECHNIQUES	16
3.1	Fast Sylvester Equation Solver for Plain Networks	16
3.2	Fast Sylvester Equation Solver for Attributed Networks	28
3.3	Novel Application: Interactive Subgraph Matching	37
CHAPTER 4	PAIRWISE ASSOCIATION WITH NEURAL TECHNIQUES	57
4.1	Sylvester Multi-Graph Neural Network	57
4.2	A Unified View of (Neural) Sylvester Equation Model	69
4.3	Novel Application: Social Recommendation with GNNs	73
CHAPTER 5	HIGH-ORDER ASSOCIATION WITH NUMERICAL TECHNIQUES	87
5.1	Background and Motivation for Multi-Way Association	87
5.2	Multi-Way Association on Plain Networks	88
5.3	Multi-Way Association on Attributed Networks	98
5.4	Experimental Evaluations	103
CHAPTER 6	HIGH-ORDER ASSOCIATION WITH NEURAL TECHNIQUES	111
6.1	Multi-Resolution Multi-Network Embedding	111
6.2	Hypergraph Representation Learning with Pre-Training	130
6.3	A Case Study: Inconsistent Variation Family Detection	148
CHAPTER 7	CONCLUSION AND FUTURE DIRECTIONS	153
7.1	Summary of Thesis Work	153
7.2	Future Directions	155
REFERENCES		157

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Recent years have witnessed tremendous growth of the network data, which is commonly generated and extracted either from various sources or different instances of the same domain. For example, there exist multiple partially overlapped academic collaboration networks collected from DBLP, Google scholar and ArnetMiner website [1]; multi-view networks can be extracted from the same social network platform (e.g. Twitter) with different categories of node/edge views [2, 3]; medium-scaled enzyme-enzyme interaction and large-scaled protein-protein interaction networks can be acquired from bioinformatic experiments [4, 5], etc. On the other hand, network mining techniques have made great strides in a variety of tasks in extensive application domains, ranging from the classic ranking problem and similarity searching task on regular/heterogeneous information networks [6, 7], to recent advances in fact checking and query answering (QA) problem on knowledge graphs [8, 9, 10], and many more.

Despite all these progress, the majority of research in network mining focuses on single simple networks, primarily focusing on pairwise node relations within the same network. However, the aforementioned network data often contains far more than pairwise node-to-node connection information within the same network. In order to effectively mine profound insights from the data, the high-order node relations should be explored and exploited. For example, in biochemical reaction networks, the effect of different sets of nodes might be more important than single node pairs [11, 12, 13]. In the study of human interaction in multi-channel or multi-view social networks, the key often lies in the relationship of collections of nodes across multiple channels/views [14, 15, 16]. However, it is often non-trivial to either adapt single network mining algorithms for multi-network mining tasks, or develop new algorithms. Specifically, there exists unique patterns in multi-networks which can not be captured only by simply applying mining techniques for single simple networks. For example, applying independent network embedding methods on individual networks results in latent node representations of separate embedding spaces, which hinders the direct comparison of such node embeddings [17, 18]. Naively using classic random walk with restart based techniques on multiple networks with anchor links can only obtain effective node proximities within well-connected individual networks [19, 20]. Simply adapting neural models such as graph autoencoder on hypergraphs fails to capture the unique high-order node associations

of hypergraphs¹ [13, 22]. Overall, it is difficult to leverage approaches directly from single simple networks onto multi-networks, and thus it calls for new algorithms for mining multi-network data.

1.2 RESEARCH CHALLENGES

Among others, we identify the following key challenges in designing new algorithms for multi-network association.

First (*problem formulation*), there are two possible methods to define multi-network associations, namely the explicit and implicit methods. On one hand, for the explicit method, we should formally define the multi-network association mathematically and develop inference algorithms for solving it. For multi-network node proximity, the pairwise explicit association can naturally be represented by a matrix. The high-order explicit association is often represented by a tensor. Specifically, how can we encode the topological node similarities as well as node/edge attribute similarities in modeling the multi-network association? How can we incorporate prior multi-network association knowledge in calculating the multi-network association? On the other hand, for the implicit method, instead of calculating the multi-network association directly, the focus is to implicitly preserve such association while solving related mining tasks (e.g. embedding, classification, etc.). There are two categories of implicit multi-network association. First, we can represent explicit multi-network association in its low-rank form, in which the explicit association is preserved by a similarity measure of the low-rank features. Second, the implicit multi-network association is obtained by learning a low-dimensional embedding via neural models. Generally speaking, the major challenge is two-fold, including (1) how to design models to capture such implicit associations in a multi-network scenario; (2) how to learn the topological as well as attribute similarities of high-order network objects?

Second (*computational complexity*), multi-network problems are usually much more complex than their single-network counterparts, in terms of the input network sizes (multiple networks vs. single network), and the complexity of network data models (e.g. hypergraphs vs. simple networks), which leads to algorithms with potentially higher time and space complexities. For instance, given a single simple network G with n nodes and m edges. Calculating the Personalized Pagerank (PPR) [23] for a given node costs $O(\#iter \cdot (m + n))$ in time complexity by the basic fixed point method. However, given two networks G_1, G_2 , calculating cross-network node proximities by IsoRank [24] costs $O(\#iter \cdot (m^2 + n^2))$ in time

¹Hypergraph can be seen as a special case (without domain networks) of the network of networks (NoN) data model, which is a multi-network data model for modeling networks of multiple granularities [21].

complexity by the basic fixed point method that is similar to the one for PPR. How can we design algorithms which can not only tackle the multi-network association tasks, but also enjoy tractable complexity on large networks?

Third (*application*), it is unclear how multi-network association-based approaches could either improve the traditional mining tasks compared with directly applying existing methods on multi-networks, and/or enable novel applications, which are inapplicable by naively adapting single-network approaches. For example, how can we collectively align multiple similar users from multiple social networks? How can we jointly recommend appropriate music styles, artists and albums to users by comprehensively considering their interactive associations? How can we interactively answer users' queries in the form of queries graphs in a large network dataset? How can we simultaneously associate network objects of different granularities (e.g., nodes/subgraphs/graphs) to enable recommending users to groups or groups to users?

With the above motivations and challenges, the goal of this Ph.D. thesis is to **design novel algorithms to infer multi-network association in order to empower various multi-network applications.**

1.3 RESEARCH TASK OVERVIEW

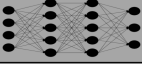
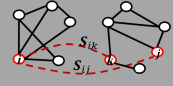
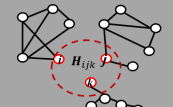
To address the challenges mentioned beforehand, we specifically explore the following four distinctive but internally correlated research problems, categorized by the pairwise/high-order association and the taxonomy of methodologies (i.e. numerical/neural techniques). The structure of this categorization can be present as a 2-by-2 table, which is shown in Table. 1.1 .

1.3.1 A Divided View of Research Tasks

Pairwise Network Association. As a sub-area of multi-network association, pairwise network association is extensively studied because of its broad applications. We will focus on numerical optimization techniques by a special category of matrix equations (i.e. Sylvester equation), and its instantiation on solving the pairwise association for various multi-network mining tasks. The neural extension of such matrix equations and two representation learning models will be further explored.

Task 1.1 - Pairwise Network Association with Numerical Techniques. Numerical methods for pairwise association represent the pairwise network association as a score matrix, whose entry equals to the weighted total support by all possible cross-network

Table 1.1: The overall scope and the taxonomy of research works in this thesis.

Methods	Numerical Techniques	Neural Techniques
Association type	$X = A_2XA_1^T + B$	
Pairwise Association	 KDD'18 Fast Sylvester equation solver KDD'17 Interactive subgraph matching	IJCAI'22 (in submission) Sylvester Multi-Graph Neural Network IJCAI'22 (in submission) Neural social recommendation
High-order Association	 KDD'21 Multi-way association	CIKM'19 Multi-resolution multi-network embedding WWW'22 (in submission) Hypergraph embedding with pretraining

matches. Based on this idea, we can formulate the problem of pairwise association as a convex optimization problem with the idea of a generalized principle of network alignment, and then concisely represent the solution of this matrix by a Sylvester equation. Moreover, we develop an efficient solver (FASTEN [25]) for this special Sylvester equation for both plain and attributed networks. FASTEN efficiently solves the Sylvester solution without approximation, and the algorithm could achieve more than 10,000 times faster compared with the traditional linear system solvers. Furthermore, we also show the applicability of the Sylvester equation in enabling novel multi-network mining tasks, such as collective network alignment, interactive subgraph matching, and so on. Specifically, we develop a family of efficient algorithm, FIRST [17], for answering query graphs with dynamic query modification from user feedback. The FIRST algorithm adopts an approximate method in solving the Sylvester equation, which could preserve more than 90% accuracy while achieve up to 16 times speed-up over baseline methods.

Task 1.2 - Pairwise Network Association with Neural Techniques. Existing neural models try to learn comparable latent node representations for multi-network tasks such as network alignment. We focus on not only comparable cross-network node embeddings, but also multi-resolution embedding which is comparable between multi-resolution network objects from different networks. Furthermore, we will elucidate the relation between our numerical techniques on pairwise network association inference (matrix equation) and neural models (e.g., via Graph Neural Networks). Specifically, we propose a multi-resolution multi-network embedding model (MRMINE [26]), for learning the embedding of network objects

from different resolutions and different networks on the same embedding space. We also develop a neural framework as an generalization for the numerical Sylvester equation (i.e., Sylvester Multi-Graph Neural Network), and show the applicability of its instantiations in geometric matrix completion (SYMGNN [27]) and social recommendation (NEMOS [28]).

High-order Network Association. As a generalization of pair-wise network association, high-order network association deals with more complex network data models (including multiple regular networks and hypergraphs), problem formulation, and corresponding algorithms. We will focus on generalizing numerical optimization by Sylvester equation to high-order association on regular networks, and several neural approaches for high-order association on hypergraphs as well as multi-network embeddings.

Task 1.3 - High-order Network Association with Numerical Techniques. Existing numerical approaches for high-order association aim to learn the multi-relations across entities of different domains, typically via tensor factorization. We resort to a generic convex optimization formulation which can be solved by a Sylvester tensor equation. We propose an efficient algorithm (SYTE [15]) for solving such special tensor equations for the high-order association solution on both plain and attributed networks. SYTE enjoys a linear time and space complexity, and is shown to be effective in a variety of multi-network mining tasks, such as multi-network alignment, multi-network node retrieval and high-order recommendation.

Task 1.4 - High-order Network Association with Neural Techniques. Capturing high-order association with latent representations by neural models is the key of this category of methods. In this thesis, we introduce a hypergraph representation learning framework (HyperGRL [29]) which leverages the strength of pre-training strategy and Graph Neural Networks (GNN). The proposed model is shown to outperform all baseline methods in hyperedge classification by up to 5.7%. Additionally, we conduct a real-world case study, in which HyperGRL is successfully applied in the inconsistent variation family detection for the Amazon selection and catalog system.

In summary, the research of this thesis boils down to design principled algorithms to solve the three aforementioned challenges for problems of pairwise and high-order network association inference and applications. Numerical and neural techniques consist of the two categories of the mining approaches in the multi-network scenario. We will elaborate the algorithms as well as the applications. Moreover, we will elucidate how these two seemingly distinctive techniques are related with each other.

1.3.2 A Unified View of Research Tasks

Unification of Pairwise and High-order Network Association. As discussed above, high-order network association is a generalization of pair-wise network association towards

more than two networks, which make it natural to unify the two areas of problems and their corresponding methods under the umbrella of the multi-network association. For the problem, the unified multi-network association problem takes the inputs of multiple attributed networks, and outputs either the explicit collective associations of nodes across input networks, or the node embeddings which implicitly encode such collective associations. For the methods, the high-order multi-network association methods could usually degenerate to pairwise association methods when the number of input networks is two. But as we will see in Chapter 3 and Chapter 5, specific problems will tackle different challenges, and it is difficult to have a one-fits-all method.

Unification of Numerical and Neural Techniques. Although the numerical and neural techniques for the multi-network association problems appears to be two separate directions, there exists a potential unified view to relate them. As shown in Table 1.1, the Sylvester Multi-Graph Neural Network [27] serves as a bridge between the blocks of numerical techniques and neural techniques for pairwise association. Specifically, the traditional Sylvester equation-based methods for explicitly calculating multi-network association can be reformulated and generalized into a neural framework, which is learnable and incorporated with downstream tasks end-to-end. We will elaborate this unified view in detail in Chapter 4.

1.3.3 Overview of the Strengths and Weaknesses of Techniques

Numerical techniques usually refer to a series of methods which use numerical approximation for mathematical analysis, such as numerous numerical methods for solving linear systems. Neural techniques usually represent a category of machine learning methods which adopt various (deep) neural network architectures in the model. Here, taking the representative numerical and neural techniques (i.e., Sylvester equation-based approaches and the Graph Neural Networks-based approaches) as examples here, we summarize the general strengths and weaknesses in Table 1.2 of the numerical and neural techniques corresponding to the taxonomy of the topics shown in Table 1.1.

1.4 THESIS ORGANIZATION

The remainders of the thesis are organized as follows. In Chapter 2, we first review the related literature in the exact same taxonomy as this thesis (Table 1.1), and then summarize the applications and limitations of the existing works. In Chapter 3, we present the fast Sylvester equation solver for both plain and attributed networks, with one novel application, namely interactive subgraph matching. In Chapter 4, we introduce our works of

Table 1.2: The strengths and weaknesses of the multi-network association techniques.

Techniques	Numerical Techniques	Neural Techniques
Strengths	<ul style="list-style-type: none"> • Theoretical strength: able to guarantee the solution existence, uniqueness, robustness, etc.; • Efficiency: efficient methods exist; No hidden states/representations; • Able to leverage long-range topological dependency. 	<ul style="list-style-type: none"> • Able to incorporate heterogeneous features; • Able to leverage non-linear relationship between input features and outputs; • Easy to be adapted to other downstream tasks end-to-end.
Weaknesses	<ul style="list-style-type: none"> • Difficult to incorporate heterogeneous features; • Unable to capture non-linear relationship between input and output; • Solution should always be adapted to a downstream task by separate learning model, but not in an end-to-end fashion. 	<ul style="list-style-type: none"> • Difficult to theoretically analyze the solution’s properties; • Developing efficient method is an open challenge; Need to save hidden states/representations; • Tend to suffer from over-smoothing when leveraging long-range dependency.

Sylvester Multi-Graph Neural Network with one novel application: social recommendation with Graph Neural Networks. In Chapter 5, we present the problem and solutions of multi-way association. In Chapter 6, we elaborate our approaches for multi-resolution multi-network embedding, as well as a hypergraph representation learning method with a case study on inconsistent variation family detection. We conclude the thesis and discuss the future directions in Chapter 7.

CHAPTER 2: LITERATURE REIVEW

In this chapter, we review the literature of the related work of the thesis topic. Following the research problems of the thesis, we divide this chapter into two categories, including the pairwise network association and the high-order network association. For each category, we further divide it into numerical techniques and neural techniques.

2.1 PAIRWISE NETWORK ASSOCIATION

Pairwise network association refers to the node associations between pairs of nodes across different networks in a multi-network scenario. In the following literature review, we will focus on the pairwise network association for two regular plain and attributed networks respectively. Various graph mining tasks are closely related to this problem such as network alignment [18, 19, 30], clustering [31, 32, 33], and matrix completion in multi-network scenarios [34, 35, 36], etc. They either explicitly or implicitly leverage pairwise network association for their tasks. We will review the representative works for each task.

2.1.1 Numerical Techniques

The core idea of the well-known IsoRank algorithm by Singh et al. [19, 24] is to represent the two-network alignment result as a cross-network pairwise association score matrix, whose entry (i, j) equals to the total support provided to it by each of the $|N(i)| * |N(j)|$ ($|N(i)|$: number of neighbors of node i) possible matches between the neighbors of i and j . In return, each node-pair (u, v) must distribute back its entire score equally among the $|N(u)| * |N(v)|$ possible matches between its neighbors. This idea could be concisely represented as a matrix equation on plain networks, which is later generalized as the topology consistency principle as one of the three principles of attributed network alignment in FINAL by Zhang et al. [30]. Multi-Network clustering also utilize the idea of pairwise association for effective clustering. Multi-Network Anchoring (MNA) algorithm by Kong et al. [37] formulates the pairwise association inference problem on heterogeneous networks as a stable matching problem between the two sets of nodes in two different networks, under one-to-one mapping constraint. MCA by Liu et al. [38] propose to use a cross-domain cluster alignment matrix for mapping cluster membership of one domain network to the other. The cross-domain cluster alignment matrix could be seen as a generalization of pairwise association matrix for node pairs. Matrix completion on multi-networks could be regarded as referring the missing pairwise

associations given the observed incomplete pairwise association matrix. Kalofolias et al. propose a low-rank solution [36] which is structured by the proximity between rows and columns that form communities to tackle the optimization problem of matrix completion. The formulation could combine both the collaborative filtering and content based filtering idea.

2.1.2 Neural Network-based Techniques

For each of the data mining tasks by numerical methods which we discussed beforehand, the neural techniques are also explored progressively. DMNE by Ni et al. [31] presents a multi-network embedding model which coordinates multiple neural network modules (one module for each input network) via a co-regularized loss. Another representative work, IONE by Liu et al. [39] aims at explicitly modeling input/output context representations so as to conserve the similarities of users with “similar” followers/followees pattern in the embedded space, and proposes a unified model for learning users’ embedding as well as user alignment via transferring of context information across networks. For clustering, DMGC by Luo et al. [33] proposes to use a autoencoder network with a minimum entropy clustering objective for jointly inferring cluster assignments and cluster associations in multi-network in an end-to-end model. Most recently, Yan et al. propose GraphAE [40] which targets at adapting the learned network representation to other networks while implicitly leveraging the cross-network pairwise associations. Recent Graph Neural Networks based approaches for matrix completion on multi-networks include [34, 41, 42, 43, 44, 45]. Other neural approaches of multi-network embedding with pairwise association for various specific application scenarios include [46, 47, 48, 49, 50, 51, 52]. Lastly, as we will review in the next two subsections, the methods that target at alignment/clustering/tensor completion on more than two networks might be degenerated to pairwise association problem in their corresponding settings.

2.2 HIGH-ORDER NETWORK ASSOCIATION

High-Order network association refers to the node associations beyond pairwise relationship, such as the node association across more than two regular networks, and the node association of hyperedges in a hypergraph. In the following literature review, we will focus on the high-order network association for multiple regular networks and hypergraphs respectively.

2.2.1 Numerical Techniques

Traditional high-order association research is often referred to as (multi-)relational learning (MRL), in a sense that the target is to learn the interaction relations across different entities of different domains [53, 54, 55, 56]. These works commonly adopt the matrix/tensor factorization approach. But they either do not explicitly model the interactions of entities as networks or implicitly model such interactions by graph regularizer, which imposing conditions to make similar entities close in their feature representations. Recently, the research on network-based multi-relational learning begins to draw attention, and is most relevant to the focus of this thesis. Representative works are briefly described as follows. TOP by Liu et al. [57] formulate the problem as a convex optimization problem, which enables transductive learning using both labeled and unlabeled tuples, and offers a scalable (linear w.r.t. input network sizes) algorithm. Li et al. [16] propose a general tensor-based optimization framework to infer the multi-relations among the entities across multiple networks in a low-rank tensor. The paper particularly studies the multi-relations between protein-protein interaction network, gene-gene relation network, and human disease-disease similarity network. Recently, Li et al. propose LowrankTLP [58], whose formulation generalizes a widely used label propagation model to the tensor product graph. The algorithm is efficient by sequentially selecting the eigen-pairs from each individual network.

For high-order network association on hypergraphs, leveraging hypergraph laplacian-based objective with a regularizer formulation solves a wide range of research problems. This method is originated from [59] by Zhou et al., in which the authors propose to formulate the hypergraph partitioning as a ranking-based convex optimization problem. The formulation and its variants with different regularizer are found to be applicable in numerous problems, such as link prediction in social networks [60], image/text retrieval [61], recommendation [62], social mining [63], bioinformatics [64], and multimedia research [12]. Meanwhile, other numerical techniques are also applied to solve various problems. For example, Zhang et al. propose a Coordinated Matrix Minimization (CMM) algorithm. It uses coordinated matrix minimization method for nonnegative matrix factorization in the adjacency space of the hypergraphs for hyperedge prediction. MetaFac by Lin et al. [65] propose to use hypergraph for modeling the relational and multi-dimensional social data for multi-relational learning, and develop a tensor factorization based method for community detection. Recently, there are also reflection on how to effectively construct hypergraph from raw data for obtaining optimal mining task performance with classic methods [11, 66, 67]. For example, Yoon et al. investigate the conditions for constructing hypergraphs with different levels of abstraction specifically for hyperlink prediction [66]. Sharma et al. find that the 2-project graph fails

when capturing high-order associations [67].

2.2.2 Neural Network-based Techniques

For high-order network association on multiple regular networks, the number of literature is limited compared with classic numerical methods, but many recent works can be regarded as implicitly utilizing high-order associations. The current approaches could be roughly divided as non-convolutional based and convolutional based. Here we review some representative works. Aimed at local network clustering, the algorithm AMRWR by Yan et al. [32] develops a Multi-Network Random Walk with Restart (MRWR) schema, which could be seen as a modified truncated random walk applied on multi-networks for discovering local clusters on a given network in association with additional networks. DeepMNE by Xue et al. [68] presents a semi-supervised autoencoder model for learning feature latent representations of nodes of multiple regular networks. CrossMNA by Chu et al. [18] leverages the cross-network information of multi-networks to refine two types of node embeddings, the inter-vector for multi-network alignment and intra-vector for other downstream network analysis tasks. LinkNBed by Trivedi et al. [69] aims at entity linkage in multiple knowledge graphs, and proposes a deep relational learning framework which learns entity and relationship representations across multiple graphs. These works implicitly preserve high-order associations as building the optimization model. High-Order association on regular networks could usually be used in the following applications, i.e. multi-network alignment [18], high-order social recommendation [70, 71], etc.

For high-order network association on hypergraphs, similar to regular network scenario, two major directions are explored in recent literature, namely the non-convolutional techniques, such as language model-inspired models (e.g. Skipgram) and graph auto-encoder-based models, and also the models based on Graph Neural Networks (GNN). Specifically, Yang et al. [72] target at location based social networks (LBSN), and first propose a random-walk-with-stay scheme to jointly sample user check-ins and social relationships. Then it learns node embeddings from the sampled (hyper)edges by preserving n-wise node proximity by a Skipgram-based model. After that, Yang et al. continue to propose LBSN2Vec++ [73] for heterogeneous hypergraph embedding in LBSNs. RNe2Vec by Shang et al. [74] follow the direction of [72] and propose a biased random walk based approach for capturing different repost behaviors in a post network. Besides all the embedding methods using Skipgram, graph auto-encoder is also explored. DHNE by Tu et al. [22] prove that the hypergraph embedding learning should conserve the intrinsic indecomposability property of hyperedges via nonlinear functions. In order to achieve this, the model utilizes an encoder-decoder module

and a non-linear module in the model architecture of hypergraph embedding learning.

On the other hand, recent years have witnessed a surge of Graph Neural Network-based neural models for mining tasks in hypergraphs in the literature. We will review a few representative works, some of which are in the frontier of unprecedented research directions. One of the earliest works that explore the applicability of GNN on hypergraph is [75] by Arya et al. It adopts the idea of multi-graph CNN model from [34] by Monti et al., which is an early multi-network convolutional model for geometric matrix completion. Later, Feng et al. proposes hypergraph neural network (HGNN) [76], which designs a hyperedge convolution operation to handle the data correlation during hypergraph representation learning. Hyper-SAGNN by Zhang et al. [77] investigates the potential of combining attention-based dynamic mechanism with a static neural network mechanism to achieve superior performance in hyperlink prediction. HyperGCN by Yadati et al. [13] present how to adapt Graph Convolutional Networks (GCN) on hypergraphs with derivation from hypergraph laplacian. Moreover, inspired by recent exploration of using GNN model on combinatorial problems on regular graphs, it tries to solve combinatorial problems on hypergraphs for the first time. Most recently, Yu et al. studies the potential of using self-training strategy, which is originally developed in natural language processing and computer vision communities, on hypergraph representation learning for social recommendation [78] for the first time. Beyond static hypergraph mining, Jiang et al. [79] demonstrate how GNN could be adapted for representation learning on dynamic hypergraphs.

As for applications, unlike classic approaches of mining hypergraphs, neural approaches only focus on a small number of real-world data mining tasks, such as hyperlink prediction [80], and recommendation [78]. Further investigation of how to boost diverse applications could be one future direction.

2.3 MULTI-NETWORK ASSOCIATION APPLICATIONS

Here, we review some representative graph mining applications which leverage the multi-network association either explicitly or implicitly. These applications are all closely related to the scope of this thesis.

2.3.1 Network Alignment

Network alignment is an important aspect of applications closely related to the multi-network association. Based on the specific problem settings, the network alignment problem can be divided into the following categories: (1) pairwise network alignment, which aims

at finding the node correspondence across two networks [30, 81, 82]; (2) collective network alignment, which targets at collectively align multiple networks [15, 83]; (3) High-Order network alignment, which maximizes the number of correspondent high-order structures, such as triangles [84]; (4) Hierarchical network alignment, which simultaneously aligns the nodes and clusters of nodes in different resolutions. Multi-level optimization and multi-resolution matrix factorization techniques have been applied to tackle this problem [85, 86]. A complete survey on network alignment research can be found in [87].

2.3.2 Social Recommendation

The related works of social recommendation can be roughly divided into two categories, the classical social recommendation methods and the recent GNN-based models.

Classical Social Recommendation Methods. *SocialMF* [88] incorporates the mechanism of trust propagation into the matrix factorization-based model, and shows its effectiveness in tackling cold-start problems. *TrustSVD* [89] extends the idea of trust propagation to leverage both explicit and implicit influence of ratings and trust in the matrix factorization model. *ContexMF* [90] proposes a probabilistic matrix factorization method to fuse the individual preference and interpersonal influence. *CNSR* [91] is one of the earliest works to introduce neural models in social recommendation. It proposes two modules, namely a social embedding and a collaborative neural recommendation part, and further combines them in a joint learning framework.

GNN-based Models. Numerous GNN-based models have been proposed. We review some of the most representative works here. *PinSage* [92] is one of the earliest works to apply GNNs on recommender systems. The model constructs convolution operations via random walks to generate embeddings of items which incorporate both graph topology and feature information. *NGCF* [93] proposes to propagate the user/item embeddings on the user-item bipartite graph for modeling the high-order connectivity. *GraphRec* [44] proposes a neural model to jointly capture the interactions and opinions in the user-item graph, in order to handle the heterogeneity issue in the user-user and user-item relation. Recently, *DiffNet* [94] and *DiffNet++* [95] propose a GNN-based model to combine the social influence diffusion with the interest diffusion, with different designs of GNN architectures. As a follow-up, *DiffNetLG* [96] models both local implicit influence of users on unobserved interpersonal relations, and global implicit influence of items broadcasted to users. *LightGCN* [97] simplifies the GCN model in recommendation to only keep the neighbor aggregation for collaborative filtering. *RecQ* [78] develops a hypergraph-based model to model the high-order user relations in social recommendation. *ESRF* [98] addresses the three commonly

observed drawbacks in practice with a deep adversarial framework.

2.3.3 Link Prediction in Hypergraphs

Among the earliest work, Li et al. [60] transform the hyperlink prediction problem as a ranking problem for hyperedges, and adopt a SimRank-like [99] method. [14] use a coordinated matrix minimization method for nonnegative matrix factorization in the adjacency space of the hypergraphs for hyperedge prediction. More recently, representation learning and GNN-based methods are developed for this direction. Tu et al. propose DHNE [100] which applies auto-encoder and deep neural networks for the representation learning of hyperedges. Hyper-SAGNN by Zhang et al. [77] combines the attention-based dynamic representations and the MLP-based static representations for hyperedge prediction. HGNN by Feng et al. [76] generalizes the convolution operation to the hypergraph by hypergraph Laplacian. Deep Hyperedges [101] by Payne jointly uses contextual and permutation-invariant node memberships of hyperedges for hyperedge classification.

2.3.4 Subgraph Matching

Various subgraph matching techniques can be found in different targeting problems. Traditional exact subgraph matching algorithms include Ullmann [102], TurboISO [103], and CFL [104]. G-Ray by Tong et al. [105] applies *RWR* (Random Walk with Restart) [106, 107] and *CePS* (CenterPiece Subgraphs) [108] idea to achieve fast inexact pattern matching for networks with node attributes. *TALE* by Tian and Patel [109] allows approximate matching and large query graphs by proposing NH-index (Neighborhood Index). *SIGMA* by Mongiovi et al. [110] defines a new effective pruning rule for inexact matching based on multi-set and multi-cover, a variant of the well known set-cover problem. It performs well in the application of query yeast and human protein complexes. More recently, *R-WAG*, *I-WAG* and *S-WAG* by Roy et al. [111] aim to return fast best-effort answer for WAG (Weighted Attributed Graphs) query by designing a hybrid index structure that incorporates weighted attributes, structural features and graph structure. *NeMa* by Khan et al. [112] proposes a heuristic approach based on defining a new definition of matching cost metric. More recently, *IncMatch* by Fan et al. presents a simulated method for incremental subgraph matching of certain patterns. Recently, G-Finder by Liu et al. [113] is proposed to leverage traditional indexing methods on inexact matching scenario.

2.4 LIMITATIONS FOR EXISTING WORKS

From a high-level perspective, the existing works on multi-network association exhibit the following three limitations. First, the learned multi-network association does not always effectively encode the multi-network topology or multi-network attributes. For example, given two attributed networks as inputs for network alignment, many existing works could not effectively combine the nodes' local topology features with their numerical features or attributes [81, 99, 114]. Consequently, the learned multi-network association or node embeddings could fall short in achieving superior performance. As another example in social recommendation, many existing works fail to leverage both the user-user social interaction and the user-item relation of interest, when learning user/item embeddings for recommendation [44, 90, 91, 92]. Second, the existing methods often suffer from high time and space complexities when dealing with large multi-networks. As we have discussed in Chapter 1, the solution space of the multi-network association problem becomes significantly larger compared with single network node association problem [25]. Therefore, the computational complexity is also much larger. Furthermore, when learning representations of multi-networks, the model architecture becomes more complex compared with single network models. For example, conducting random walks on multi-network data for unsupervised embedding becomes highly non-trivial compared with random walks on single simple networks, in which the random walks are well-defined [20, 26]. Third, the existing methods are often difficult to be adapted for new multi-network applications. For example, the pairwise network association methods are difficult to be applied directly for high-order association problems for multi-networks [29].

CHAPTER 3: PAIRWISE ASSOCIATION WITH NUMERICAL TECHNIQUES

The Sylvester equation is the cornerstone of many multi-network mining tasks, such as network alignment, subgraph matching, graph kernel, and node similarity. In this chapter, we introduce our works on the solution of the Sylvester equation and its applications, including (1) a family of fast Sylvester equation solvers for plain networks where no attributes are available [25]; (2) a family of fast Sylvester equation solvers for attributed networks [25]; and (3) a novel application of Sylvester equation for multi-network mining, namely interactive subgraph matching [17].

3.1 FAST SYLVESTER EQUATION SOLVER FOR PLAIN NETWORKS

How can we link users from different social network sites (e.g., Facebook, Twitter, LinkedIn, etc.)? How can we integrate different tissue-specific protein-protein interaction (PPI) networks together to prioritize candidate genes? How to predict the toxicity of chemical molecules by comparing their three-dimensional structure? How can we detect suspicious transaction patterns (e.g., money-laundering ring) in the financial network given a template pattern/query? The Sylvester equation [115], defined over the adjacency matrices of the input networks, provides a powerful and unifying primitive for a variety of key graph mining tasks, including network alignment [116], graph kernel [117], node similarity [118], subgraph matching [17], etc. Figure 3.1 presents an illustrative example of using Sylvester equation for graph mining.

A major limitation of Sylvester equation lies in the high computational complexity. For graphs with n nodes and m edges, the straight-forward solver is $O(n^6)$ in time and $O(m^2)$ in space. Much effort has been devoted to speed up the computation for solving Sylvester equation. Nonetheless, state-of-the-art methods for plain graphs require a complexity that is at least $O(mn + n^2)$ [30]. The complexity for solving Sylvester equation would be even more intensified when the input graphs have accompanying node attributes. For example, it would add an additional $O(l)$ (l is the number of node attribute values) to the time complexity of conjugate gradient descent solver. Some recent approximate algorithms try to reduce this complexity by matrix low-rank approximation. Nonetheless, it still requires $O(n^2)$, even with such an approximation. Table 3.1 summarizes the time and space complexity of the existing methods for solving Sylvester equations. Consequently, most of the existing Sylvester equation solvers can only handle graphs with up to tens of thousands of nodes.

To address this issue, we design a family of Krylov subspace based algorithms (FASTEN)

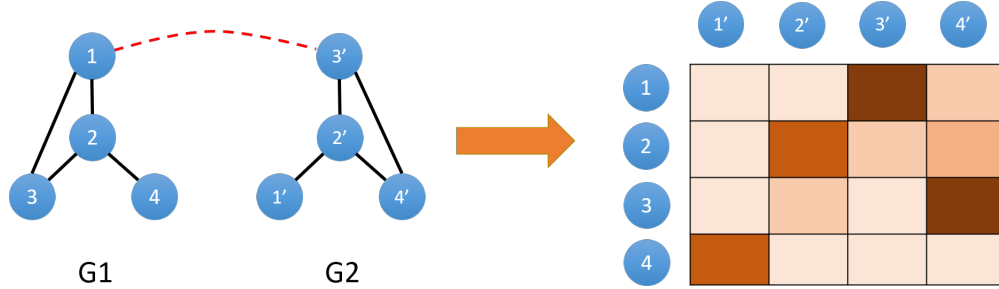


Figure 3.1: An illustrative example of network alignment by Sylvester equation [30, 119]. Left: the input plain graphs with the dashed line indicating the preference matrix \mathbf{B} . Right: the solution matrix \mathbf{X} of the corresponding Sylvester equation $\mathbf{X} = \mathbf{A}_1\mathbf{X}\mathbf{A}_2 + \mathbf{B}$ gives the cross-network node similarity, where \mathbf{A}_1 and \mathbf{A}_2 are the adjacency matrices of the input graphs; and a darker square on the right represents higher cross-network node similarity. Best viewed in color.

to speed up and scale up the computation of Sylvester equation for graph mining. The key idea of the designed methods is to project the original corresponding linear system onto a Kronecker Krylov subspace. By doing so, we are able to obtain an $O(n)$ factor reduction in time complexity and $O(m^2/n^2)$ factor reduction in space complexity compared to normal Krylov subspace based Sylvester equation solver for plain graphs [115]. Building upon that, we seek to further reduce the complexity. Here, our key observation is that the preference matrix in the Sylvester equation is often low-rank (e.g. Figure 3.1), which implies that the solution matrix itself must have low-rank structure. This, together with some additional optimization (e.g., exploiting the linearity by using the block-diagonal structure of the solution matrix for the attributed graphs), helps further reduces the complexity of the designed method to be *linear* in both time and space, without approximation error. Table 3.1 (shaded parts) summarizes the time and space complexities of the FASTEN algorithms. The main contributions of this work are as follows:

- **Novel Algorithms.** We design a family of efficient and accurate Sylvester equation solver for graph mining tasks, with and without node attribute.
- **Proof and Analysis.** We provide theoretic analysis of the developed algorithms in terms of the accuracy and complexity.
- **Empirical Evaluations.** We perform extensive experimental evaluations on a diverse set of real networks with a variety of graph mining tasks, which demonstrate that our methods (1) are up to 10,000 \times faster than the Conjugate Gradient method, the best known competitor that output exact solution, and (2) scale up to million-node graphs.

Table 3.1: Complexity Summary and Comparison. r (the rank of input graphs), l (number of node attributes) and k (subspace size) are much smaller compared with m and n . Some small constants are omitted for clarity.

Algorithm	Attributed(Y/N)	Exact(Y/N)	Time	Space
<i>FP</i> [117, 120]	Y	Y	$O(n^3)$	$O(m^2)$
<i>CG</i> [117, 120]	Y	Y	$O(n^3)$	$O(m^2)$
<i>Sylv.</i> [117, 120]	Y	Y	$O(n^3)$	$O(m^2)$
<i>ARK</i> [121]	Y	N	$O(n^2)$	$O(n^2)$
<i>Cheetah</i> [122]	Y	N	$O(rn^2)$	$O(n^2)$
<i>NI-Sim</i> [119]	N	N	$O(n^2)$	$O(r^2n^2)$
<i>FINAL-P</i> [30]	N	Y	$O(mn + n^2)$	$O(n^2)$
<i>FINAL-NE</i> [30]	Y	Y	$O(lmn + ln^2)$	$O(n^2)$
<i>FINAL-N+</i> [30]	Y	N	$O(n^2)$	$O(n^2)$
<i>FASTEN-P</i>	N	Y	$O(kn^2)$	$O(n^2)$
<i>FASTEN-P+</i>	N	Y	$O(km + kn)$	$O(m + kn)$
<i>FASTEN-N</i>	Y	Y	$O(mn/l + kn^2/l)$	$O(m/l + n^2)$
<i>FASTEN-N+</i>	Y	Y	$O(km + k^2ln)$	$O(m + kln)$

3.1.1 Problem Definition

The main symbols and notations used in this work are summarized in Table 3.2. The calligraphic letters \mathcal{G}_1 and \mathcal{G}_2 represent two attributed graphs. The uppercase bold letters \mathbf{A}_1 and \mathbf{A}_2 represent the $n \times n$ adjacency matrices and the uppercase bold letters \mathbf{N}_1 and \mathbf{N}_2 represent the node attribute matrices. \mathbf{N}_1 and \mathbf{N}_2 are diagonal matrices in which $\mathbf{N}_1^j(a, a) = 1$ if the node a in graph \mathcal{G}_1 has node attribute j and otherwise it is zero. The uppercase bold letter \mathbf{N} without subscript or superscript is the combined node attribute matrix of two input graphs. $\mathbf{N} = \sum_{j=1}^l \mathbf{N}_1^j \otimes \mathbf{N}_2^j$ where l is the number of node attributes. We use uppercase bold letter \mathbf{D} as the combined diagonal degree matrix of the two input graphs. For a matrix (e.g., \mathbf{X}), we vectorize it in the column order to obtain its equivalent vector representation (e.g., $\mathbf{x} = \text{vec}(\mathbf{X})$). For an attributed network, we index its nodes by the corresponding attributes in the adjacency matrix, i.e., all the nodes with the same attribute are consecutive rows/columns in the adjacency matrix, and their induced subgraph is a block of the adjacency matrix with consecutive indices. For example, for \mathcal{G}_1 in Figure 3.2, node-1 and node-2 are the first two rows/columns in the adjacency matrix \mathbf{A}_1 since they share the same node attribute (blue diamond); and node-3 and node-4 are the last two rows/columns in \mathbf{A}_1 since they share the same node attribute (green hexagon). This will simplify the description of the developed algorithms.

Sylvester equation for graph mining. For the completeness, we present a brief review of Sylvester equations and their applications to graph mining. For more details, please

Table 3.2: Symbols and Definition

Symbols	Definition
$\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{N}_1\}$	an attributed graph
\mathbf{N}	combined node attribute matrix of input graphs
\mathbf{R}, \mathbf{r}	residual matrix and residual vector
$\mathbf{H}_1, \mathbf{H}_2$	$k \times k$ Hessenberg matrices
$\mathbf{D}_1, \mathbf{D}_2$	diagonal degree matrices
\mathbf{I}	an identity matrix
\mathbf{B}	preference matrix in the Sylvester equation
$\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$	Krylov subspace with dimension k
k	Krylov subspace size $k \ll n$
α	the parameter $0 < \alpha < 1$
l	the number of node attribute
$b = \text{vec}(B)$	vectorize a matrix B in the column order
$[\mathbf{A}, \mathbf{B}]$	concatenate two matrices in a row
$[\mathbf{A}; \mathbf{B}]$	concatenate two matrices in a column
$\text{diag}(\mathbf{A}_1, \dots, \mathbf{A}_i)$	diagonalize i matrices
\otimes	Kronecker product
$\text{trace}(\cdot)$	trace of a matrix
$\ \cdot\ _F$	Frobenius norm

refer to [30, 117, 119]. For graphs without attributes, let $\mathbf{A}_1 \leftarrow \alpha^{1/2} \mathbf{D}_1^{-1/2} \mathbf{A}_1 \mathbf{D}_1^{-1/2}$ and $\mathbf{A}_2 \leftarrow \alpha^{1/2} \mathbf{D}_2^{-1/2} \mathbf{A}_2 \mathbf{D}_2^{-1/2}$ be the normalized adjacency matrices of two input graphs, and \mathbf{B} be the preference matrix. We have the following Sylvester equation [30].

$$\mathbf{X} - \mathbf{A}_2 \mathbf{X} \mathbf{A}_1^T = \mathbf{B} \quad (3.1)$$

By the Kronecker product property, we have the equivalent linear system of Equation (3.1) as follows.

$$(\mathbf{I} - \mathbf{W})\mathbf{x} = \mathbf{b} \quad (3.2)$$

where $\mathbf{W} = \mathbf{A}_1 \otimes \mathbf{A}_2$, $\mathbf{b} = \text{vec}(\mathbf{B})$ and $\mathbf{x} = \text{vec}(\mathbf{X})$. We assume that \mathbf{A}_1 and \mathbf{A}_2 are of the same size $n \times n$. For example, for the two input graphs in Figure 3.1, \mathbf{A}_1 and \mathbf{A}_2 are 4×4 adjacency matrices of \mathcal{G}_1 and \mathcal{G}_2 respectively, and \mathbf{B} is the preference matrix to reflect the prior knowledge (the dashed line). The entries of the solution matrix \mathbf{X} indicate the similarity between a node pair across two input graphs.

When the input graphs have attributes on nodes, the \mathbf{W} matrix in Equation (3.2) becomes $\mathbf{W} = \mathbf{D}^{-1/2} [\mathbf{N}(\mathbf{A}_1 \otimes \mathbf{A}_2) \mathbf{N}] \mathbf{D}^{-1/2}$. To simplify the notation, let $\mathbf{A}_1^{(ij)} \leftarrow \alpha^{1/2} \mathbf{D}_1^{-1/2} \mathbf{N}_1^i \mathbf{A}_1 \mathbf{N}_1^j \mathbf{D}_1^{-1/2}$, $\mathbf{A}_2^{(ij)} \leftarrow \alpha^{1/2} \mathbf{D}_2^{-1/2} \mathbf{N}_2^i \mathbf{A}_2 \mathbf{N}_2^j \mathbf{D}_2^{-1/2}$ be the attributed, normalized adjacency matrices. We can see that $\mathbf{A}_1^{(ij)}$ is of the same size of \mathbf{A}_1 , but it is ‘filtered’ by the corresponding attributes.

In other words, $\mathbf{A}_1^{(ij)}$ only contains links in \mathbf{A}_1 from nodes with attribute i to nodes with attributes j . In this case, Equation (3.2) becomes:

$$[\mathbf{I} - \sum_{i=1}^l \sum_{j=1}^l (\mathbf{A}_1^{(ij)} \otimes \mathbf{A}_2^{(ij)})] \mathbf{x} = \mathbf{b} \quad (3.3)$$

Again, by the Kronecker product property, we have the following equivalent Sylvester equation of Equation (3.3):

$$\mathbf{X} - \sum_{i=1}^l \sum_{j=1}^l \mathbf{A}_2^{(ij)} \mathbf{X} (\mathbf{A}_1^{(ij)})^T = \mathbf{B} \quad (3.4)$$

For the ease of description of the developed algorithms, we also use a block-matrix representation. Take \mathbf{A}_1 for an example, we have $\mathbf{A}_1 = [\mathbf{A}_1^{ij}]_{i,j=1,\dots,l}$, where \mathbf{A}_1^{ij} is a block of matrix \mathbf{A}_1 from rows of attribute i to columns of attribute j . Note the subtle difference between \mathbf{A}_1^{ij} and $\mathbf{A}_1^{(ij)}$: they have different sizes and we can verify that $\mathbf{A}_1 = \sum_{i,j=1}^l \mathbf{A}_1^{(ij)}$. We use a similar representation for other matrices in Equation (3.4), i.e., $\mathbf{A}_2 = [\mathbf{A}_2^{ij}]_{i,j=1,\dots,l}$, $\mathbf{B} = [\mathbf{B}^{ij}]_{i,j=1,\dots,l}$, and $\mathbf{X} = [\mathbf{X}^{ij}]_{i,j=1,\dots,l}$.

Figure 3.2 presents an illustrative example of attributed Sylvester equation. \mathbf{A}_1 and \mathbf{A}_2 are two 4×4 adjacency matrices, and \mathbf{N}_1 and \mathbf{N}_2 are the node attribute matrices of \mathcal{G}_1 and \mathcal{G}_2 . \mathbf{B} represents the prior knowledge (the dashed line). Each entry of the solution \mathbf{X} indicates a similarity score between a node pair across \mathcal{G}_1 and \mathcal{G}_2 . Although both $\mathbf{A}_1^{(11)}$ and \mathbf{A}_1^{11} correspond to links between node-1 and node-2 (since both nodes share the first attribute, i.e., blue diamond), their sizes are different: $\mathbf{A}_1^{(11)}$ is of 4×4 whereas \mathbf{A}_1^{11} is of 2×2 .

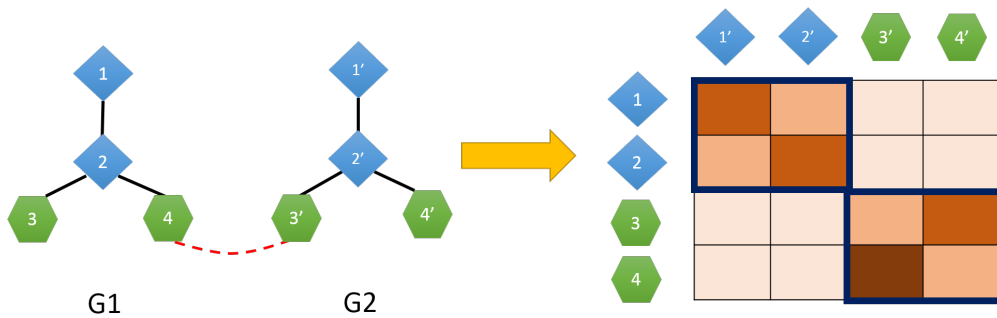


Figure 3.2: An illustrative example of attributed network alignment by Sylvester equation [30, 119]. Left: the input graphs with node attributes (colors and shapes). Dashed line: preference matrix. Right: the solution matrix \mathbf{X} of Equation (3.4), representing cross-network node similarity. A darker square represents a higher similarity value. Best viewed in color.

The Sylvester equations defined in Equation (3.1) and Equation (3.4), together with their equivalent linear systems (Equation (3.2) and Equation (3.3)) provide a very powerful tool for many graph mining tasks. For example, the solution matrix \mathbf{X} indicates the cross-network node similarity, which can be directly used for the task of network alignment; by aggregating the solution matrix \mathbf{X} (e.g., by a weighted linear summation over the entries in \mathbf{X}), it measures the similarity between the two input graphs [121]; if one of the two input graph is a small query graph, the solution matrix \mathbf{X} becomes the basis for (interactive) subgraph matching [17]; if the two input graphs are identical without node attributes, the corresponding Sylvester equation degenerates to *SimRank* and thus its solution matrix \mathbf{X} measures the node similarity [119].

However, as mentioned earlier, a major bottleneck lies in the high computational complexity. In the next two sections, we present our solutions to speed up and scale up the computation of Sylvester equations, which are divided into two parts based on whether or not the input graphs are attributed. See Table 3.1 for a summary and comparison.

Krylov subspace methods for linear systems. A classic method for solving a linear system $\mathbf{Ax} = \mathbf{b}$ is via Krylov subspace methods [115]. It first generates an orthogonal basis of the its Krylov subspace with an initial residual vector \mathbf{r}_0 , denoted by $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$, where $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$, and \mathbf{x}_0 is an initial solution. Then it iteratively updates the residual vector and the corresponding solution vector \mathbf{x} over the Krylov subspace formed equation [115] Compared with the alternative methods, e.g., (conjugate) gradient descent, the major advantage of Krylov method lies in the ability to project the original system onto a subspace with a much smaller dimension/size, which can be in turn solved very efficiently.

3.1.2 FASTEN-P(+): Fast Algorithms for Plain Networks.

Here, we address the Sylvester equation for plain graphs without node attributes, i.e., Equation (3.1) and Equation (3.2). We start with the key ideas and intuition behind the developed algorithms, and then present the detailed algorithms (FASTEN-P and FASTEN-P+), followed by some analysis in terms of the accuracy and complexity.

Intuition and Key Ideas. Let us first highlight the key ideas and intuition behind the two developed algorithms, using the example in Figure 3.1. First, if we directly apply the standard Krylov subspace methods to solve Equation (3.2), we would need to generate the orthonormal basis of $\mathcal{K}_{k^2}(\mathbf{I} - \mathbf{A}_1 \otimes \mathbf{A}_2, \mathbf{r}_0)$ or $\mathcal{K}_{k^2}(\mathbf{A}_1 \otimes \mathbf{A}_2, \mathbf{r}_0)$, which requires a complexity as high as $O(n^4)$ in both time and space. To avoid such a high polynomial complexity, our

first key idea is to represent the Krylov subspace of $\mathbf{I} - \mathbf{A}_1 \otimes \mathbf{A}_2$ as well as conduct the subsequently computation to update the residual/solution vectors *indirectly* by the Krylov spaces of the two input graphs. Taking Figure 3.1 for an example, where the two adjacency matrices are both 4×4 . The Krylov subspace of the corresponding Sylvester equation is in \mathbb{R}^{16} . As will shown in the developed FASTEN-P algorithm, we will decompose it into the Kronecker product of two Krylov subspace in \mathbb{R}^4 , which will largely reduce the time and space cost (FASTEN-P).

Second, notice that the solution matrix \mathbf{X} itself is of size $n \times n$. Thus, if we compute and store it in a straight-forward way, it still needs $O(n^2)$ complexity. Here, the key observation is that the preference matrix \mathbf{B} often has a low-rank structure. For the example in Figure 3.1, the preference matrix \mathbf{B} only has two non-zero entries ($\mathbf{B}(3, 4)$ and $\mathbf{B}(4, 3)$), making itself a rank-2 matrix. This observation is crucial as it allows us to perform all the intermediate computation as well as to represent the solution matrix \mathbf{X} in an indirect way, leading to a *linear* complexity (FASTEN-P+).

FASTEN-P Algorithm. Let \mathbf{D}_1 and \mathbf{D}_2 be the diagonal degree matrices of the adjacency matrices of the two input graphs \mathbf{A}_1 and \mathbf{A}_2 respectively. We normalize the adjacency matrices as $\mathbf{A}_1 \leftarrow \alpha^{1/2} \mathbf{D}_1^{-1/2} \mathbf{A}_1 \mathbf{D}_1^{-1/2}$, and $\mathbf{A}_2 \leftarrow \alpha^{1/2} \mathbf{D}_2^{-1/2} \mathbf{A}_2 \mathbf{D}_2^{-1/2}$, where $0 < \alpha < 1$ is a regularization parameter.

We first show how to represent $\mathcal{K}_{k^2}(\mathbf{A}_1 \otimes \mathbf{A}_2, \mathbf{r}_0)$ *indirectly* in the form $\mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$. To be specific, let $\mathbf{V}_i = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_i]$, $\mathbf{W}_j = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_j]$ for $i \in \{k, k+1\}$, and $j \in \{k, k+1\}$, where $\{\mathbf{v}_i\}_{i=1}^k$ and $\{\mathbf{w}_j\}_{j=1}^k$ are orthonormal basis of $\mathcal{K}_k(\mathbf{A}_1, \mathbf{g})$ and $\mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$, respectively. Let \mathbf{H}_1 , \mathbf{H}_2 and $\tilde{\mathbf{H}}_1$, $\tilde{\mathbf{H}}_2$ be the Hessenberg and Hessenberg-like matrices generated by the Arnoldi process¹ [115]. We have that

$$\mathbf{H}_1 = \mathbf{V}_k^T \mathbf{A}_1 \mathbf{V}_k, \mathbf{H}_2 = \mathbf{W}_k^T \mathbf{A}_2 \mathbf{W}_k \quad (3.5a)$$

$$\tilde{\mathbf{H}}_1 = \mathbf{V}_{k+1}^T \mathbf{A}_1 \mathbf{V}_k, \tilde{\mathbf{H}}_2 = \mathbf{W}_{k+1}^T \mathbf{A}_2 \mathbf{W}_k \quad (3.5b)$$

\mathbf{H}_1 , \mathbf{H}_2 are $k \times k$ and $\tilde{\mathbf{H}}_1$, $\tilde{\mathbf{H}}_2$ are $(k+1) \times k$. Notice that k is often much smaller than n (i.e., $k \ll n$). Therefore the size of the above Hessenberg and Hessenberg-like matrices is small. We can prove that $\mathbf{V}_k \otimes \mathbf{W}_k$ forms the orthonormal basis of $\mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$ (see the details in the proof of Theorem 3.1).

Next, we show how to update the residual vector \mathbf{r} and the solution vector \mathbf{x} using the indirect representation of $\mathcal{K}_{k^2}(\mathbf{A}_1 \otimes \mathbf{A}_2, \mathbf{r}_0)$. Let \mathbf{x}_0 be an initial solution of Equation (3.2).

¹Note that if \mathbf{A}_1 and \mathbf{A}_2 are symmetric (undirected graphs), Lanczos algorithm can be applied. We use Arnoldi process for generality.

The initial residual vector is:

$$\mathbf{r}_0 = \mathbf{b} - (\mathbf{I} - \alpha \mathbf{W})\mathbf{x}_0 \quad (3.6)$$

To obtain a new solution $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{z}_0$, we want $\mathbf{z}_0 \in \mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$, i.e.,

$$\mathbf{z}_0 = (\mathbf{V}_k \otimes \mathbf{W}_k)\mathbf{y} \quad (3.7)$$

for an unknown vector $\mathbf{y} \in \mathbb{R}^{k^2}$. The new residual is:

$$\mathbf{r}_1 = \mathbf{r}_0 - [(\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{V}_k \otimes \mathbf{W}_k)\mathbf{y} - (\mathbf{V}_k \otimes \mathbf{W}_k)\mathbf{y}] \quad (3.8)$$

Based on Equation (3.5a), we seek to minimize the residual:

$$\begin{aligned} \|\mathbf{r}_1\|_2 &= \min_{\mathbf{y} \in \mathbb{R}^{k^2}} \|\mathbf{r}_0 - (\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{V}_k \otimes \mathbf{W}_k)\mathbf{y} + (\mathbf{V}_k \otimes \mathbf{W}_k)\mathbf{y}\|_2 \\ &= \min_{\mathbf{y} \in \mathbb{R}^{k^2}} \|\mathbf{r}_0 - (\mathbf{A}_1 \mathbf{V}_k) \otimes (\mathbf{A}_2 \mathbf{W}_k)\mathbf{y} + (\mathbf{V}_k \otimes \mathbf{W}_k)\mathbf{y}\|_2 \\ &= \min_{\mathbf{y} \in \mathbb{R}^{k^2}} \|\mathbf{r}_0 - (\mathbf{V}_{k+1} \tilde{\mathbf{H}}_1) \otimes (\mathbf{W}_{k+1} \tilde{\mathbf{H}}_2)\mathbf{y} + (\mathbf{V}_k \otimes \mathbf{W}_k)\mathbf{y}\|_2 \\ &= \min_{\mathbf{y} \in \mathbb{R}^{k^2}} \|(\mathbf{V}_{k+1} \otimes \mathbf{W}_{k+1})[(\mathbf{V}_{k+1} \otimes \mathbf{W}_{k+1})^T \mathbf{r}_0 - (\tilde{\mathbf{H}}_1 \otimes \tilde{\mathbf{H}}_2)\mathbf{y} + (\mathbf{I}_{k+1,k} \otimes \mathbf{I}_{k+1,k})\mathbf{y}]\|_2 \\ &= \min_{\mathbf{y} \in \mathbb{R}^{k^2}} \|(\mathbf{V}_{k+1} \otimes \mathbf{W}_{k+1})^T \mathbf{r}_0 - (\tilde{\mathbf{H}}_1 \otimes \tilde{\mathbf{H}}_2)\mathbf{y} + (\mathbf{I}_{k+1,k} \otimes \mathbf{I}_{k+1,k})\mathbf{y}\|_2 \end{aligned} \quad (3.9)$$

where $\mathbf{I}_{k,k+1} = [\delta_{i,j}]_{1 \leq i \leq k+1, 1 \leq j \leq k}$ and $\delta_{i,j}$ is the Kronecker δ -function. Equation (5.14) can be solved by:

$$\min_{\mathbf{Y} \in \mathbb{R}^{k^2}} \|(\mathbf{W}_{k+1}^T \mathbf{R}_0 \mathbf{V}_{k+1} - \tilde{\mathbf{H}}_2 \mathbf{Y} \tilde{\mathbf{H}}_1^T + \mathbf{I}_{k+1,k} \mathbf{Y} \mathbf{I}_{k+1,k}^T)\|_F \quad (3.10)$$

It can be proved that the least square problem in equation (3.10) can be solved by the following linear system (see details in the proof of Theorem 3.1):

$$L(\mathbf{Y}) = \mathbf{C} \quad (3.11)$$

where $L(\mathbf{Y}) = \tilde{\mathbf{H}}_2^T \tilde{\mathbf{H}}_2 \mathbf{Y} \tilde{\mathbf{H}}_1^T \tilde{\mathbf{H}}_1 - \mathbf{H}_2^T \mathbf{Y} \mathbf{H}_1 - \mathbf{H}_2 \mathbf{Y} \mathbf{H}_1^T + \mathbf{Y}$, and $\mathbf{C} = \tilde{\mathbf{H}}_2^T \mathbf{W}_{k+1}^T \mathbf{R}_0 \mathbf{V}_{k+1} \tilde{\mathbf{H}}_1 - \mathbf{W}_k^T \mathbf{R}_0 \mathbf{V}_k$. Notice that the size of both \mathbf{Y} and \mathbf{C} are $k \times k$, so the dimension of Equation (3.11) is typically small and we can solve it by Global Conjugate method [123].

Another subtle issue is how to choose the initial vectors \mathbf{g} and \mathbf{f} for the Arnoldi Process. We use a procedure in [124] to choose \mathbf{f} and \mathbf{g} to guarantee that $\mathbf{r}_0 \in \mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$ as follows. If $\|\mathbf{R}_0\|_1 \leq \|\mathbf{R}_0\|_\infty$, we choose \mathbf{f} as the column of \mathbf{R}_0 with the largest l_2 norm and $\mathbf{g} = \mathbf{R}_0^T \mathbf{f} / \|\mathbf{f}\|_2^2$; otherwise, we choose \mathbf{g} as the row of \mathbf{R}_0 with the largest l_2 norm and

$$\mathbf{f} = \mathbf{R}_0 \mathbf{g} / \|\mathbf{g}\|_2^2.$$

Putting everything together, we have the overall algorithm for solving the Equation (3.1) in Algorithm 3.1.

Algorithm 3.1 FASTEN-P

Input: Normalized adjacency matrices \mathbf{A}_1 and \mathbf{A}_2 , tolerance parameter $\epsilon > 0$, Krylov subspace size $k > 0$, preference matrix \mathbf{B} ;

Output: The solution \mathbf{X} of Equation (3.1).

- 1: Initialize \mathbf{X} and the residual matrix \mathbf{R} ;
 - 2: **while** $\|\mathbf{R}\|_F > \epsilon$ **do**
 - 3: Choose Arnoldi vectors $\mathbf{g} \in \mathbb{R}^n$ and $\mathbf{f} \in \mathbb{R}^n$;
 - 4: Apply Arnoldi Process on \mathbf{A}_1 , \mathbf{A}_2 and obtain $\tilde{\mathbf{H}}_1$, $\tilde{\mathbf{H}}_2$, \mathbf{V}_k , \mathbf{V}_{k+1} , \mathbf{W}_k , \mathbf{W}_{k+1} ;
 - 5: Use Global Conjugate Gradient method for the linear system $L(\mathbf{Y}) = \mathbf{C}$;
 - 6: Update solution $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{V}_k \mathbf{Y} \mathbf{W}_k^T$;
 - 7: Update residual $\mathbf{R} \leftarrow \mathbf{R} - \mathbf{V}_{k+1} \tilde{\mathbf{H}}_1 \mathbf{Y} \tilde{\mathbf{H}}_2^T \mathbf{W}_{k+1}^T + \mathbf{V}_k \mathbf{Y} \mathbf{W}_k^T$;
 - 8: **end while**
-

The first line is the initialization of the solution matrix and the residual. Line 2 to 8 are the outer loop, while line 5 is the inner loop which uses the Global Conjugate Gradient method. Line 6 (Equation (3.7)) updates the solution matrix \mathbf{X} at each iteration. Line 7 (Equation (3.8) and Equation (5.14)) updates the residual matrix \mathbf{R} at each iteration.

FASTEN-P+ Algorithm. To further reduce the time and space complexity of FASTEN-P, we explore the low-rank structure of the preference matrix \mathbf{B} of Equation (3.1). Firstly, it is common in many graph mining tasks that the matrix \mathbf{B} of Equation (3.1) has a low-rank structure. For example, in network alignment, anchor links are often sparse (e.g. in Figure 3.1, only 1 anchor link is given). If the prior knowledge of anchor links is unknown, matrix \mathbf{B} becomes a uniform matrix, which means it is rank-1. Secondly, it turns out that the low-rank structure of the preference matrix \mathbf{B} would imply the solution matrix \mathbf{X} must have a low-rank block matrix structure (see the detailed analysis and proof in Lemma 3.1). This allows us to implicitly represent both the residual matrix and the solution matrix, which leads to a *linear* complexity.

The improved algorithm (FASTEN-P+) is summarized in Algorithm 3.2. As we can see the residual \mathbf{R} is implicitly represented by the multiplication of \mathbf{U}_1 and \mathbf{U}_2 , and the solution \mathbf{X} is implicitly represented by the multiplication of \mathbf{M} and \mathbf{N} . The residual is updated at each iteration in line 9 and the solution is updated in line 7. Due to the implicit representation of the initial residual matrix, we need a slightly different procedure to choose the Arnoldi vector in line 4. Given the implicit representation of residual by \mathbf{U}_1 and \mathbf{U}_2 , let $\mathbf{r}_1 = \mathbf{e}^T \mathbf{U}_1 \mathbf{U}_2$, $\mathbf{r}_2 = \mathbf{U}_1 \mathbf{U}_2 \mathbf{e}$ (\mathbf{e} is an all-one vector), and let i_1 and i_2 be the indexes of

the largest entries in \mathbf{r}_1 and \mathbf{r}_2 . If $\max(\mathbf{r}_1) \geq \max(\mathbf{r}_2)$, we choose \mathbf{f} as $\mathbf{U}_1\mathbf{U}_2(:, i_1)$, and then $\mathbf{g} = \mathbf{U}_2^T\mathbf{U}_1^T\mathbf{f}/|\mathbf{f}|_2^2$; otherwise, we set \mathbf{g} as $\mathbf{U}_2^T\mathbf{U}_1(i_2, :)^T$, and then $\mathbf{f} = \mathbf{U}_1\mathbf{U}_2\mathbf{g}/|\mathbf{g}|_2^2$.

Algorithm 3.2 FASTEN-P+

Input: Normalized adjacency matrices \mathbf{A}_1 and \mathbf{A}_2 , tolerance parameter $\epsilon > 0$, Krylov subspace size $k > 0$, preference matrix \mathbf{B} ;

Output: The implicit representation for the solution matrix \mathbf{X} of Equation (3.1): \mathbf{P} and \mathbf{Q} .

- 1: Initialize \mathbf{P}, \mathbf{Q} . Set \mathbf{e} be an all-one vector.;
 - 2: Let $\mathbf{U}_1 \leftarrow \mathbf{B}(:, 1)$, $\mathbf{U}_2 \leftarrow \mathbf{e}$;
 - 3: **while** $\text{trace}(\mathbf{U}_2^T(\mathbf{U}_1^T\mathbf{U}_1)\mathbf{U}_2) > \epsilon$ **do**
 - 4: Choose Arnoldi vectors $\mathbf{g} \in \mathbb{R}^n$ and $\mathbf{f} \in \mathbb{R}^n$;
 - 5: Apply Arnoldi Process on \mathbf{A}_1 and \mathbf{A}_2 to obtain $\tilde{\mathbf{H}}_1, \tilde{\mathbf{H}}_2, \mathbf{V}_k, \mathbf{V}_{k+1}, \mathbf{W}_k, \mathbf{W}_{k+1}$;
 - 6: Use Global Conjugate Gradient method for $L(\mathbf{Y}) = \mathbf{C}$;
 - 7: Construct $\mathbf{P} \leftarrow [\mathbf{P}, \mathbf{V}_k\mathbf{Y}]$, $\mathbf{Q} \leftarrow [\mathbf{Q}, \mathbf{W}_k^T]$;
 - 8: $\mathbf{L}_2 \leftarrow \mathbf{V}_{k+1}\tilde{\mathbf{H}}_1\mathbf{Y}\tilde{\mathbf{H}}_2^T$, $\mathbf{P}_2 \leftarrow \mathbf{W}_{k+1}^T$, $\mathbf{L}_3 \leftarrow \mathbf{V}_k\mathbf{Y}$, $\mathbf{P}_3 \leftarrow \mathbf{W}_k^T$;
 - 9: Construct $\mathbf{U}_1 \leftarrow [\mathbf{U}_1, \mathbf{L}_2, \mathbf{L}_3]$, $\mathbf{U}_2 \leftarrow [\mathbf{U}_2^T, \mathbf{P}_2^T, \mathbf{P}_3^T]^T$;
 - 10: **end while**
 - 11: Return \mathbf{P}, \mathbf{Q} .
-

From line 3 to line 10 are the outer loop where the algorithm updates $\mathbf{P}, \mathbf{Q}, \mathbf{U}_1, \mathbf{U}_2$ and checks stopping condition. Line 6 is the inner loop of Global Conjugate Gradient method.

We remark that the low-rank representation for the solution matrix \mathbf{X} in Algorithm 3.2 bears subtle difference from [30, 121, 125], which *approximate* the input adjacency matrices with an inevitable approximation error. In contrast, the low-rank representation of the solution matrix in the developed FASTEN-N is due to the low-rank structure of the preference matrix \mathbf{B} , regardless whether or not \mathbf{A}_1 and \mathbf{A}_2 are low-rank. As such, it does not introduce any approximation error.

Proofs and Analysis. We provide the proofs and analysis of the proposed algorithms as follows.

Correctness. The correctness of FASTEN-P is summarized in Theorem 3.1, which says the matrix \mathbf{X} by Algorithm 3.1 finds the exact solution of Equation (3.1).²

Theorem 3.1 (Correctness of FASTEN-P). The matrix \mathbf{X} by Algorithm 3.1 is the exact solution of Equation (3.1) w.r.t the tolerance ϵ .

Proof. First, we prove that $\mathbf{V}_k \otimes \mathbf{W}_k$ is the orthonormal basis of $\mathcal{K}_k(\mathbf{A}_1, g) \otimes \mathcal{K}_k(\mathbf{A}_2, f)$.

²Following the convention in scientific computing, we say a solution \mathbf{X} is exact w.r.t. a tolerance parameter ϵ if the Frobenius norm of the difference between \mathbf{X} and the solution by the direct method is less than ϵ . Throughout the paper, ϵ is set to be a very small number, e.g., $\epsilon = 10^{-7}$.

The Hessenberg matrix $\mathbf{H}_1 \otimes \mathbf{H}_2$ and $\mathbf{A}_1 \otimes \mathbf{A}_2$ satisfy the following relationship

$$\begin{aligned}\mathbf{H}_1 \otimes \mathbf{H}_2 &= (\mathbf{V}_k^T \mathbf{A}_1 \mathbf{V}_k) \otimes (\mathbf{W}_k^T \mathbf{A}_2 \mathbf{W}_k) \\ &= (\mathbf{V}_k^T \otimes \mathbf{W}_k^T)(\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{V}_k \otimes \mathbf{W}_k) \\ &= (\mathbf{V}_k \otimes \mathbf{W}_k)^T(\mathbf{A}_1 \otimes \mathbf{A}_2)(\mathbf{V}_k \otimes \mathbf{W}_k)\end{aligned}\tag{3.12}$$

Therefore, $\mathbf{V}_k \otimes \mathbf{W}_k$ forms the orthonormal basis of $\mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$.

We then show that the least square problem with regard to \mathbf{Y} in Equation (3.10) can be solved by a normal Equation (3.11).

We define the following linear operation: $\Phi(\mathbf{Y}) = \tilde{\mathbf{H}}_2 \mathbf{Y} \tilde{\mathbf{H}}_1^T - \mathbf{I}_{k+1,k} \mathbf{Y} \mathbf{I}_{k+1,k}^T$. Then, we have that $\|\mathbf{R}_1\|_F = \|\mathbf{W}_{k+1}^T \mathbf{R}_0 \mathbf{V}_{k+1} - \Phi(\mathbf{Y})\|_F$ is minimized if and only if the following condition holds [124]:

$$\Phi^T(\mathbf{W}_{k+1}^T \mathbf{R}_0 \mathbf{V}_{k+1} - \Phi(\mathbf{Y})) = 0\tag{3.13}$$

where $\Phi(\mathbf{Y})^T$ is the joint linear operation of $\Phi(\mathbf{Y})$. Therefore, we have that

$$\tilde{\mathbf{H}}_2^T(\mathbf{W}_{k+1}^T \mathbf{R}_0 \mathbf{V}_{k+1} - \Phi(\mathbf{Y}))\tilde{\mathbf{H}}_1 - \mathbf{I}_{k+1,k}^T(\mathbf{W}_{k+1}^T \mathbf{R}_0 \mathbf{V}_{k+1} - \Phi(\mathbf{Y}))\mathbf{I}_{k+1,k} = 0\tag{3.14}$$

which is equivalent to Equation (3.11).

Finally, we prove that Algorithm 3.1 could give the exact solution of Equation (3.2).

For an arbitrary vector $\mathbf{v} \in \mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$, we have that $(\mathbf{I} - \mathbf{A}_1 \otimes \mathbf{A}_2)\mathbf{v}$ also belongs to $\mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$. This is because $(\mathbf{A}_1 \otimes \mathbf{A}_2)\mathbf{v} \in \mathcal{K}_k(\mathbf{A}_1, \mathbf{g}) \otimes \mathcal{K}_k(\mathbf{A}_2, \mathbf{f})$ when $\tilde{\mathbf{H}}_1(k+1, k) = \tilde{\mathbf{H}}_2(k+1, k) = 0$ before the convergence.

Therefore, the Kronecker subspace is invariant under matrix $\mathbf{I} - \mathbf{A}_1 \otimes \mathbf{A}_2$. According to [115], this is a sufficient condition for the solution obtained from any (oblique or orthogonal) projection method onto the Kronecker subspace to be exact. Thus, FASTEN-P gives the exact solution of Equation (3.1) or its equivalent linear system (Equation (3.2)), which completes the proof. QED.

In order to prove the correctness of FASTEN-P+, we first give the following lemma, which says that the low-rank structure of the preference matrix \mathbf{B} implies that the solution matrix \mathbf{X} must be low-rank as well.

Lemma 3.1 (Low-rank structure of the Sylvester equation). If the preference matrix \mathbf{B} in Equation (3.1) is of low-rank r , then the rank of the solution matrix \mathbf{X} of the Equation (3.1) is upper-bounded by pkr , where p is the number of iterations in the outer loop in Algorithm 3.2.

Proof. Suppose the rank- r preference matrix \mathbf{B} can be represented as $\mathbf{B} = \mathbf{B}_1\mathbf{B}_2^\top$, where $\mathbf{B}_1, \mathbf{B}_2 \in \mathbb{R}^{n \times r}$. Every iteration of the calculation for \mathbf{P} and \mathbf{Q} in Algorithm 3.2 concatenates matrices $\mathbf{V}_k\mathbf{Y}$ and \mathbf{W}_k , which has rank k , to the implicit solution matrices. After p iterations, the ranks of \mathbf{P} and \mathbf{Q} are at most pk . QED.

Based on Lemma 3.1 and following a similar process as the proof for Theorem 3.1, we can prove that Algorithm 3.2 also gives the exact solution of Equation (3.1).

Lemma 3.2 (Correctness of FASTEN-P+). If \mathbf{P} and \mathbf{Q} are the two matrices returned by Algorithm 3.2, matrix $\mathbf{X} = \mathbf{P}\mathbf{Q}$ is the exact solution of Equation (3.1) w.r.t the tolerance ϵ .

Proof. From Algorithm 3.1, the solution \mathbf{X} is calculated as the summation of the elements in $\{\mathbf{V}_k^{(i)}\mathbf{Y}^{(i)}(\mathbf{W}_k^{(i)})^\top\}_{i=1}^p$ where i is the index of iterations and p is the number of total iterations. From the matrix structure of \mathbf{P} and \mathbf{Q} , we can see that $\mathbf{P}\mathbf{Q} = \mathbf{V}_k^{(1)}\mathbf{Y}^{(1)}(\mathbf{W}_k^{(1)})^\top + \dots + \mathbf{V}_k^{(p)}\mathbf{Y}^{(p)}(\mathbf{W}_k^{(p)})^\top$ which is equal to the solution of Algorithm 3.1. According to Theorem 3.1, the correctness of Algorithm 3.2 also holds. QED.

Efficiency. The time and space complexity of FASTEN-P and FASTEN-P+ are summarized in Lemma 3.3. We can see that the developed FASTEN-P has a quadratic complexity in both time and space, which is already better than state-of-the-art algorithms for solving Sylvester equations on plain graphs - they either produce an inexact solution or require a super-quadratic time complexity (See Table 3.1 for a comparison). Furthermore, the developed FASTEN-P+ has a linear complexity in both time and space.

Lemma 3.3 (Complexity of FASTEN-P and FASTEN-P+). The time and space complexity of FASTEN-P are $O((2k+4)pn^2)$ and $O(n^2)$ respectively. The time and space complexity of FASTEN-P+ are $O(kp(m+(r+2k+1)n))$ and $O(m+(r+2k+1)n)$ respectively, where p is the number of iterations in the outer loop and r is the rank of \mathbf{B} .

Proof. First, for FASTEN-P Algorithm, as we can see from Algorithm 3.1 and Algorithm 3.6, the main computation lies in the update of solution matrix, residual matrix, and k steps of Arnoldi iteration. Specifically, for updating the solution matrix and residual matrix in each iteration, the time complexity is $O(4n^2)$. For the Arnoldi algorithm on \mathbf{A}_1 and \mathbf{A}_2 in each iteration, the time complexity is $O(2kn^2)$. Overall, the time complexity for FASTEN-P for p iterations is $O((2k+4)pn^2)$. For space complexity, since we need to store the intermediate solution matrix and the residual matrix, the space complexity is $O(n^2)$. Second, similarly for FASTEN-P+ Algorithm, the main computation lies in updating $\mathbf{P}, \mathbf{Q}, \mathbf{U}_1, \mathbf{U}_2$ matrices, which cost $O((r+2k+1)nk)$. In this case, the Arnoldi process costs $O(km)$. Overall, for the Algorithm 3.2, the time complexity is $O(kp(m+(r+2k+1)n))$. For time complexity,

because of the implicit solution \mathbf{P}, \mathbf{Q} and residual matrices $\mathbf{U}_1, \mathbf{U}_2$ in Algorithm 3.2, the space complexity is $O(m + (r + 2k + 1)n)$. QED.

3.2 FAST SYLVESTER EQUATION SOLVER FOR ATTRIBUTED NETWORKS

In this section, we present the attributed Sylvester equation solvers (FASTEN-N and FASTEN-N+) for Equation (3.3) and Equation (3.4).

3.2.1 FASTEN-N(+)

We start by introducing the intuition and key ideas, and then present the detailed algorithms, followed by some analysis in terms of the accuracy and complexity.

Intuition and Key Ideas. First, we cannot directly apply FASTEN-P or FASTEN-P+ to solve Equation (3.3) and (3.4). The main difficulty lies in the summation of Kronecker products on the left side of Equation (3.3), which should be first approximated by a single Kronecker product and itself could take $O(n^3)$ in time (nearest Kronecker product) [117]. To address this issue, the key observation is that the solution matrix \mathbf{X} for the attributed Sylvester equation (i.e., Equation (3.4)) has a block-diagonal structure. This allows us to decompose the original Sylvester equations into *a set of* inter-correlated small-scaled Sylvester equations, which can be in turn solved by block coordinate descent methods (FASTEN-N). Second, by exploring the low-rank structure of the solution matrix (as we did in the FASTEN-P+ algorithm), we will be able to obtain a linear algorithm to solve Equation (3.4). Let us explain this by the example in Figure 3.2. We can rewrite its attributed Sylvester equation as follows (Equation (5.11)). It can be seen that we only need to solve two inter-correlated Sylvester equations w.r.t two 2×2 diagonal blocks (\mathbf{X}^{11} and \mathbf{X}^{22}), which can be in turn solved by an efficient *block coordinate descent (BCD)* [126] method. For the two off-diagonal blocks (\mathbf{X}^{12} and \mathbf{X}^{21}), they are the same as the corresponding blocks in the preference matrix \mathbf{B} .

$$\mathbf{X}^{11} - [\mathbf{A}_2^{11} \mathbf{X}^{11} (\mathbf{A}_1^{11})^T + \mathbf{A}_2^{12} \mathbf{X}^{22} (\mathbf{A}_1^{12})^T] = \mathbf{B}^{11} \quad (3.15a)$$

$$\mathbf{X}^{22} - [\mathbf{A}_2^{21} \mathbf{X}^{11} (\mathbf{A}_1^{21})^T + \mathbf{A}_2^{22} \mathbf{X}^{22} (\mathbf{A}_1^{22})^T] = \mathbf{B}^{22} \quad (3.15b)$$

$$\mathbf{X}^{12} = \mathbf{B}^{12} \quad (3.15c)$$

$$\mathbf{X}^{21} = \mathbf{B}^{21} \quad (3.15d)$$

FASTEN-N Algorithm. In the general case, Equation (3.4) can be decomposed into a group of inter-correlated equations of block variables:

$$\mathbf{X}^{ii} - \sum_{q=1}^l \mathbf{A}_2^{iq} \mathbf{X}^{qq} (\mathbf{A}_1^{iq})^T = \mathbf{B}^{ii} \quad (3.16a)$$

$$\mathbf{X}^{ij} = \mathbf{B}^{ij} \quad (3.16b)$$

where $1 \leq i, j \leq l, i \neq j$. Since the off-diagonal blocks \mathbf{X}^{ij} are the same as the corresponding blocks in \mathbf{B} , we will focus on solving Equation (3.16a) in the developed algorithm, using a *BCD* method. The goal is to minimize the overall residual, namely the residual of Equation (3.4). In each iteration, one diagonal block variables will be updated with other blocks fixed. Let $\tilde{\mathbf{B}}_i = \mathbf{B}^{ii} + \sum_{j \neq i}^l \mathbf{A}_2^{ij} \mathbf{X}^{jj} (\mathbf{A}_1^{ij})^T$. We will solve the following equation in the i -th iteration in order to update \mathbf{X}^{ii} :

$$\mathbf{X}^{ii} - \mathbf{A}_2^{ii} \mathbf{X}^{ii} (\mathbf{A}_1^{ii})^T = \tilde{\mathbf{B}}_i \quad (3.17)$$

If we only treat \mathbf{X}^{ii} as variables with all other diagonal blocks fixed, Equation (3.17) is a Sylvester equation *without* attributes, and thus can be efficiently solved by the developed FASTEN-P algorithm.

Algorithm 3.3 FASTEN-N

Input: Normalized adjacency matrices \mathbf{A}_1 and \mathbf{A}_2 , node attribute matrices \mathbf{N}_1 and \mathbf{N}_2 , preference matrix \mathbf{B} , tolerance parameter $\epsilon > 0$, Krylov subspace size $k > 0$, number of node attribute l ;

Output: The solution \mathbf{X} of Equation (3.4).

- 1: Initialize each diagonal block variable $\mathbf{X}^{ii}, \forall 1 \leq i \leq l$, residual matrix \mathbf{R} ;
 - 2: Construct block matrices $\mathbf{A}_1^{ij}, \mathbf{A}_2^{ij}, \mathbf{B}^{ij}, \mathbf{B}^{ij}, \forall 1 \leq i, j \leq l$ by the node attribute matrices $\mathbf{N}_1, \mathbf{N}_2$;
 - 3: **while** $\|\mathbf{R}\|_F > \epsilon$ **do**
 - 4: **for** $i = 1, \dots, l$ **do**
 - 5: Compute $\tilde{\mathbf{B}}_i = \mathbf{B}^{ii} + \sum_{j \neq i}^l \mathbf{A}_2^{ij} \mathbf{X}^{jj} (\mathbf{A}_1^{ij})^T$;
 - 6: Apply Algorithm 3.1 on Equation (3.17) to obtain \mathbf{X}^{ii} ;
 - 7: **end for**
 - 8: Let $\mathbf{R} \leftarrow \sum_{j=1}^l (\mathbf{B}^{jj} - \mathbf{X}^{jj}) + \sum_{i=1}^l \sum_{j=1}^l \mathbf{A}_2^{ij} \mathbf{X}^{jj} (\mathbf{A}_1^{ij})^T$;
 - 9: **end while**
-

The developed FASTEN-N is summarized in the algorithm 3.3. Line 1 and 2 initialize the diagonal block variables, and the block matrices used in the decomposed set of equations. Line 3 to line 9 are the outer loop which uses the *BCD* method and checks the stopping condition. Line 6 is the inner loop of Algorithm 3.1. Line 8 updates the overall residual of Equation (3.4).

FASTEN-N+ Algorithm. In order to further reduce the time and space complexity of FASTEN-N, we explore a similar strategy as in FASTEN-P+ by the low-rank structure of the preference matrix \mathbf{B} . Here, we further represent each block matrix of \mathbf{B} in its low-rank form, which allows to implicitly represent the solution block matrices \mathbf{X}^{ii} in the low-rank forms when solving the Equation group (3.16a). The developed algorithm FASTEN-N+ is summarized in Algorithm 3.4. Like FASTEN-N, it also uses the block coordinate descent method. A key difference between FASTEN-N and FASTEN-N+ lies in that, instead of calculating $\tilde{\mathbf{B}}_i$ directly, FASTEN-N+ represents it by two matrices $\tilde{\mathbf{B}}_i^1 = [\mathbf{B}^{ii}(:, 1), \mathbf{A}_2^{ij} \mathbf{P}_i]$ and $\tilde{\mathbf{B}}_i^2 = [\mathbf{e}; \mathbf{Q}_i \mathbf{A}_1^{ij}]$, $\forall 1 \leq i \leq l$, where \mathbf{e} is an all-one vector, and $\mathbf{P}_i, \mathbf{Q}_i$ are the implicit representation of solution \mathbf{X}^{ii} .

Algorithm 3.4 FASTEN-N+

Input: Normalized adjacency matrices \mathbf{A}_1 and \mathbf{A}_2 , node attribute matrices \mathbf{N}_1 and \mathbf{N}_2 , preference matrix \mathbf{B} , tolerance parameter $\epsilon > 0$, subspace size $k > 0$;

Output: The implicit solution $\mathbf{P}_i, \mathbf{Q}_i, \forall 1 \leq i \leq l$ of Equation 3.17.

- 1: Initialize $\mathbf{P}_i, \mathbf{Q}_i, \forall 1 \leq i \leq l$. Set \mathbf{e} be an all-one vector;
 - 2: Construct block matrices $\mathbf{A}_1^{ij}, \mathbf{A}_2^{ij}, \mathbf{B}^{ij}, \mathbf{B}^{ij}, \forall 1 \leq i, j \leq l$ by the node attribute matrices \mathbf{N}_1 and \mathbf{N}_2 ;
 - 3: Let $\mathbf{U}_1^i \leftarrow \mathbf{B}^{ii}(:, 1), \mathbf{U}_2^i \leftarrow \mathbf{e}$;
 - 4: **while** $\sum_i \text{trace}((\mathbf{U}_2^i)^T ((\mathbf{U}_1^i)^T \mathbf{U}_1^i) \mathbf{U}_2^i) > \epsilon$ **do**
 - 5: **for** $i = 1, \dots, l$ **do**
 - 6: $\tilde{\mathbf{B}}_i^1 \leftarrow [\mathbf{U}_1^i, \mathbf{A}_2^{ij} \mathbf{P}_i], \tilde{\mathbf{B}}_i^2 \leftarrow [\mathbf{U}_2^i; \mathbf{Q}_i \mathbf{A}_1^{ij}], \forall 1 \leq i \leq l$;
 - 7: Apply Algorithm 3.2 on Equation (3.17) to obtain $\mathbf{P}_i, \mathbf{Q}_i$;
 - 8: **end for**
 - 9: $\mathbf{U}_1^i \leftarrow [\mathbf{U}_1^i, -\mathbf{P}_i, \mathbf{A}_2^{ij} \mathbf{P}_j], \mathbf{U}_2^i \leftarrow [\mathbf{U}_2^i; -\mathbf{Q}_i; \mathbf{Q}_j (\mathbf{A}_1^{ij})^T], \forall 1 \leq i, j \leq l$;
 - 10: **end while**
 - 11: Return \mathbf{P}_i and $\mathbf{Q}_i, (i = 1, \dots, l)$.
-

In Algorithm 3.4, line 4 to line 10 are the outer loop which uses the *BCD* method and checks the stopping condition, and line 7 is the inner loop of Algorithm 3.2. In line 6, the revised preference matrix $\tilde{\mathbf{B}}_i$ in the equation group (3.16a) is represented by two low-rank matrices. In line 9, the residual of each equation in the equation group is indirectly represented as well.

Proofs and Analysis.

Correctness. The correctness of the developed FASTEN-N and FASTEN-N+ is summarized in Theorem 3.2, which says that both algorithms find the exact solution of Equation (3.4).

Theorem 3.2 (Correctness of FASTEN-N). The solution matrix \mathbf{X} by Algorithm 3.3 is the exact solution of Equation (3.4) w.r.t the tolerance ϵ . If \mathbf{P}_i and $\mathbf{Q}_i, (i = 1, \dots, l)$ are the

matrices returned by Algorithm 3.4, matrix $\mathbf{X} = \text{diag}(\mathbf{P}_1\mathbf{Q}_1, \dots, \mathbf{P}_l\mathbf{Q}_l) + \sum_{i \neq j}^l \mathbf{B}^{(ij)}$ is the exact solution of Equation (3.4) w.r.t the tolerance ϵ .

Proof. The proof of Theorem 3.2 is two-fold. First we prove that the Equation (3.4) can actually be decomposed to the Equation group (3.16a) and (3.16b). Second, if all block variables are solved in the Equation group (3.16a), their summation is equal to the solution of Equation (3.4). First, in Equation (3.4), let $\mathbf{T} = \sum_{i=1}^l \sum_{j=1}^l \mathbf{A}_2^{ij} \mathbf{X} (\mathbf{A}_1^{ij})^T$. Unfold $\mathbf{A}_2^{ij} = \mathbf{N}_2^i \mathbf{A}_2 \mathbf{N}_2^j$, $\mathbf{A}_1^{ij} = \mathbf{N}_1^i \mathbf{A}_1 \mathbf{N}_1^j$ with the definition of attribute matrices \mathbf{N}_1 and \mathbf{N}_2 . So, $\mathbf{T} = \sum_{i=1}^l \mathbf{N}_2^i \mathbf{A}_2 (\sum_{j=1}^l \mathbf{N}_2^j \mathbf{X} \mathbf{N}_1^j) \mathbf{A}_1 \mathbf{N}_1^i$.

Because $\sum_{j=1}^l \mathbf{N}_1^j = \mathbf{I}$, $\sum_{j=1}^l \mathbf{N}_2^j = \mathbf{I}$, we can write \mathbf{X} as $\mathbf{X} = (\sum_{j=1}^l \mathbf{N}_1^j) \mathbf{X} (\sum_{j=1}^l \mathbf{N}_2^j) = \sum_{i=j}^l \mathbf{N}_2^j \mathbf{X} \mathbf{N}_1^i + \sum_{i \neq j}^l \mathbf{N}_2^j \mathbf{N}_1^i$. Let $\mathbf{X}_1 = \sum_{i=j}^l \mathbf{N}_2^j \mathbf{X} \mathbf{N}_1^i$, $\mathbf{X}_2 = \sum_{i \neq j}^l \mathbf{N}_2^j \mathbf{N}_1^i$. Therefore the Equation (3.4) can be written as:

$$\mathbf{X}_1 + \mathbf{X}_2 - \sum_{i=1}^l \mathbf{N}_2^i \mathbf{A}_2 \mathbf{X}_1 \mathbf{A}_1 \mathbf{N}_1^i = \mathbf{B}_1 + \mathbf{B}_2 \quad (3.18)$$

and can be further decomposed to:

$$\begin{cases} \mathbf{X}_1 - \sum_{i=1}^l \mathbf{N}_2^i \mathbf{A}_2 \mathbf{X}_1 \mathbf{A}_1 \mathbf{N}_1^i = \mathbf{B}_1 & (3.19a) \\ \mathbf{X}_2 = \mathbf{B}_2 & (3.19b) \end{cases}$$

where $\mathbf{B}_1 = \sum_{i=j}^l \mathbf{N}_2^j \mathbf{B} \mathbf{N}_1^i$, $\mathbf{B}_2 = \sum_{i \neq j}^l \mathbf{N}_2^j \mathbf{B} \mathbf{N}_1^i$, then $\mathbf{B} = \mathbf{B}_1 + \mathbf{B}_2$. Note that $\mathbf{X}_1 = \sum_{j=1}^l \mathbf{X}^{jj} = \sum_{j=1}^l \mathbf{N}_2^j \mathbf{X}^{jj} \mathbf{N}_1^j$, where $\mathbf{X}^{jj} = \mathbf{N}_2^j \mathbf{X} \mathbf{N}_1^j$. Equation (3.19a) can be further decomposed to $\sum_{j=1}^l \mathbf{X}^{jj} - \sum_{i=1}^l \mathbf{A}_2^{ij} \sum_{j=1}^l \mathbf{X}^{jj} (\mathbf{A}_1^{ij})^T = \mathbf{B}_1$, which can be decomposed to Equation (3.16a) and each \mathbf{X}^{jj} can be seen as a variable. Since Equation (3.19b) is also equal to Equation (3.16b), thus the Equation (3.4) can be decomposed to Equation group (3.16a) and (3.16b).

Because taking the summation of left and right side of the equations (3.16a) and (3.16b) and we would have Equation (3.4). From previous section, FASTEN-P gives exact solution for each subequation. If all block variables are solved in the Equation group (3.16a) and (3.16b), we can have the solution of the original Equation (3.4).

Following a similar process as the proof for Theorem 3.2, we can show that Algorithm 3.4 also gives the exact solution of Equation (3.4). QED.

Lemma 3.4 (Correctness of FASTEN-N+). If \mathbf{M} and \mathbf{N} are the two matrices returned by Algorithm FASTEN-N+, matrix $\mathbf{X} = \mathbf{M}\mathbf{N}$ is the exact solution of Equation (3.4).

Proof. Similar to Lemma 3.2, since the FASTEN-N+ uses the FASTEN-P+ algorithm in

the inner iteration of Algorithm 3.4, the solution for diagonal block variant i is equal to the product of \mathbf{P}_i and \mathbf{Q}_i . When each diagonal block is constructed by the product of \mathbf{P}_i and \mathbf{Q}_i in this way, the overall solution is thus equal to the solution of FASTEN-N. QED.

Efficiency. The time and space complexity of FASTEN-N and FASTEN-N+ are summarized in Lemma 3.5. We can see that the developed FASTEN-N has a quadratic complexity in both time and space. Furthermore, the developed FASTEN-N+ has a linear complexity in both time and space (See Table 3.1 for comparison).

Lemma 3.5 (Complexity of FASTEN-N and FASTEN-N+). The time and space complexity of FASTEN-N are $O(p_2mn/l + (2k + 4)p_2n^2/l)$ and $O(n^2 + m/l)$ respectively. The time and space complexity of FASTEN-N+ are $O((p_1km + p_1k(r + lk + k + 1)n)p_2l)$ and $O(m + (r + kp_1(l - 1))n)$, respectively. p_1 is the number of iterations of FASTEN-N or FASTEN-N+ in the inner loop, and p_2 is the number of iterations of the outer loop, and r is the rank of the preference matrix \mathbf{B} .

Proof. First, for FASTEN-N algorithm, the main computation lies in the construction of $\tilde{\mathbf{B}}_i$ in the outer loop, which costs $O(mn/l)$. For the inner loop, it costs $O((2k + 4)n^2/l)$. Overall, for the Algorithm 3.3, the time complexity is $O(p_2mn/l + (2k + 4)p_2n^2/l)$. The main space consumption lies in the residual matrix \mathbf{R} and all $\tilde{\mathbf{B}}_i$, which in combine cost $O(n^2 + m/l)$. Second, for FASTEN-N+ algorithm, the main computation lies in the norm of the residual in the low-rank form, and the construction of $\tilde{\mathbf{B}}_i^1, \tilde{\mathbf{B}}_i^2, \tilde{\mathbf{U}}_i^1, \tilde{\mathbf{U}}_i^2$, which cost $O((p_1km + p_1k(r + lk + k + 1)n)p_2l)$ overall. For space, due to the low-rank implicit representation of the solution blocks and the residual by $\tilde{\mathbf{B}}_i^1, \tilde{\mathbf{B}}_i^2, \tilde{\mathbf{U}}_i^1, \tilde{\mathbf{U}}_i^2$, the space complexity is $O(m + (r + kp_1(l - 1))n)$ overall. QED.

3.2.2 Details of Assistant Algorithms

The Global Conjugate Gradient method and the Modified Arnoldi Iteration are summarized in Algorithm 3.5 and 3.6.

3.2.3 Experimental Results

In this section, we present the experimental results. The experiments are designed to evaluate the efficiency, the effectiveness and the parameter sensitivity of the developed family of algorithms (FASTEN).

Experimental Setup. We first describe the experimental setups as follows.

Algorithm 3.5 Global Conjugate Gradient for Equation (3.11)

Input: The right-side matrix \mathbf{C} ;

Output: The solution matrix \mathbf{Y} for Equation (3.11).

Initialize solution matrix \mathbf{Y}_0 , $\mathbf{R}_0 = \mathbf{C} - L(\mathbf{Y}_0)$, $\mathbf{P}_0 = \mathbf{R}_0$;

```
for  $j = 0, 1, \dots, j_{max}$  do
   $\alpha_j = \langle \mathbf{R}_j, \mathbf{R}_j \rangle_F / \langle L(\mathbf{P}_j), \mathbf{P}_j \rangle_F$ ;
   $\mathbf{Y}_{j+1} = \mathbf{Y}_j + \alpha_j \mathbf{P}_j$ ;
   $\mathbf{R}_{j+1} = \mathbf{R}_j - \alpha_j L(\mathbf{P}_j)$ ;
   $\beta_j = \langle \mathbf{R}_{j+1}, \mathbf{R}_{j+1} \rangle_F / \langle \mathbf{R}_j, \mathbf{R}_j \rangle_F$ ;
   $\mathbf{P}_{j+1} = \mathbf{R}_{j+1} + \beta_j \mathbf{P}_j$ ;
end for
```

Algorithm 3.6 Modified Arnoldi Iteration

Input: A vector \mathbf{v} , matrix \mathbf{A} , the subspace size k ;

Output: The orthonormal basis $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k]$, and the $k + 1$ by k Hessenberg-like matrix $\tilde{\mathbf{H}}$.

Initialize $\mathbf{v}_1 = \mathbf{v} / \|\mathbf{v}\|_2$;

```
for  $j = 0, 1, \dots, k$  do
   $\mathbf{w} = \mathbf{A}\mathbf{v}_j$ ;
  for  $i = 1, 2, \dots, j$  do
     $h_{i,j} = \mathbf{v}_i^T \mathbf{w}$ ;
     $\mathbf{w} = \mathbf{w} - h_{i,j} \mathbf{v}_i$ ;
  end for
   $h_{j+1,j} = \|\mathbf{w}\|_2$ ;
  if  $h_{j+1,j} = 0$  then Stop;
  else
     $\mathbf{v}_{j+1} = \mathbf{w} / h_{j+1,j}$ ;
  end if
end for
```

Datasets. We evaluate the developed algorithms on five real-world datasets [127], which are summarized in Table 3.3.

Table 3.3: Datasets Summary

Dataset Name	Category	# of Nodes	# of Edges
<i>DBLP</i>	Co-authorship	9,143	16,338
<i>Flickr</i>	User relationship	12,974	16,149
<i>LastFm</i>	User relationship	15,436	32,638
<i>AMiner</i>	Academic network	1,274,360	4,756,194
<i>LinkedIn</i>	Social network	6,726,290	19,360,690

- *DBLP*: This is a co-authorship network with each node representing an author. Each

author is assigned one node attribute vector of the number of publications in 29 major conferences [128].

- *Flickr*: This is a network of friends on the image and video hosting website *Flickr*. Node attribute vector is constructed from users’ profile information (e.g. age, gender, location, etc.) [129].
- *LastFm*: Collected in 2013, this is the following network of users on the music website *LastFm* [129]. A detailed profile of users is provided. The node attribute vector is also constructed from users’ profile information.
- *AMiner*: AMiner dataset represents the academic social network. Undirected edges represent co-authorship and the node attribute vector is extracted from the number of published papers [129].
- *LinkedIn*: The graph of *LinkedIn* dataset is from users’ connections in the social network *LinkedIn*. The node attribute vector is constructed from users’ profile information (e.g. age, gender, occupation, etc.)

Comparison Methods. We compare the developed methods against four baseline methods, the *Fixed Point* method (*FP*) [120], the *Conjugate Gradient Method* (*CG*) [120], *FINAL-P+* [30], and *FINAL-N+* [30]. According to [117], *CG* is best known method in terms of efficiency to obtain the exact solution of the Sylvester equation studied in this work. *FP* is widely used in solving Sylvester equation and linear system. *FINAL-P+* and *FINAL-N+* are two recent approximate methods that solve the Sylvester equation on plain graph and attributed graph respectively [30]. We set the rank of the input networks to 2 in both *FINAL-P+* and *FINAL-N+* to ensure a fast computation, and the runtime of these two methods with a higher rank would be longer than the results reported below. We terminate an algorithm if it does not finish within 3,000 seconds.

Repeatability. All datasets are publicly available. We will release the code of our developed algorithms upon the publication of the paper. All experiments are performed on a server with 64 Intel(R) Xeon(R) CPU cores at 2.00 GHz and 1.51 TB RAM. The operating system is Red Hat Enterprise Linux Server release 6.9. All codes are written in MATLAB R2017a using a single thread.

Efficiency. We next present the efficiency of the developed algorithms.

Speedup. We first evaluate how much speedup the developed algorithms can achieve over baseline methods, on seven real-world datasets, including two subsets of *AMiner* with 25K and 100K nodes respectively. The results are presented in Figure 3.3. As we can see

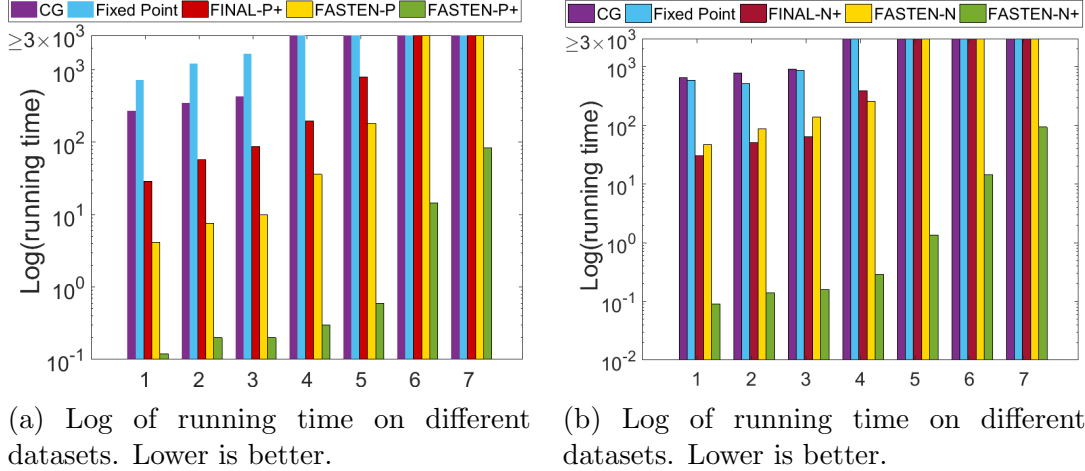


Figure 3.3: Efficiency comparison on plain and attributed networks. Best viewed in color. Datasets: 1: *DBLP*, 2: *Flickr*, 3: *LastFm*, 4: *Aminer* with 25K nodes, 5: *Aminer* with 100K nodes, 6: *Aminer*, and 7: *LinkedIn*.

from Figure 3.3(a), the developed FASTEN-P already outperforms all the baselines on all datasets even including the approximate method *FINAL-P+*. The developed FASTEN-P+ outperforms all baseline methods as well as FASTEN-P by a large margin. When it is applied on the largest dataset *LinkedIn*, the running time of FASTEN-P+ is less than 100 seconds; whereas all the other methods cannot finish within 3,000 seconds. On *AMiner* (with 25K nodes) dataset, FASTEN-P+ is more than $10,000\times$ faster than *CG* (3,000+ seconds vs. 0.3 seconds).

On the attributed networks (Figure 3.3(b)), FASTEN-N outperforms *CG* and *FP*, and its running time is close to *FINAL-N+* although the latter produces an approximate solution. The developed FASTEN-P+ outperforms all baseline methods as well as FASTEN-N by a large margin. On *AMiner* (with 25K nodes) dataset, FASTEN-N+ is more than $10,700\times$ faster than *CG*. (3,000+ seconds vs. 0.28 seconds)

Scalability. The scalability experiments are conducted on the largest *Aminer* dataset. We run experiments on graphs with different size (ranging from 5K to 1.2M nodes) 100 times, and report the average running time. The results are presented in Figure 3.9. We can see that all the baseline methods are at least quadratic w.r.t the number of nodes in graphs, and could not finish within 3,000 seconds for graphs with more than 100,000 nodes. Both the developed FASTEN-P+ and FASTEN-N+ scale linearly to million-node graphs.

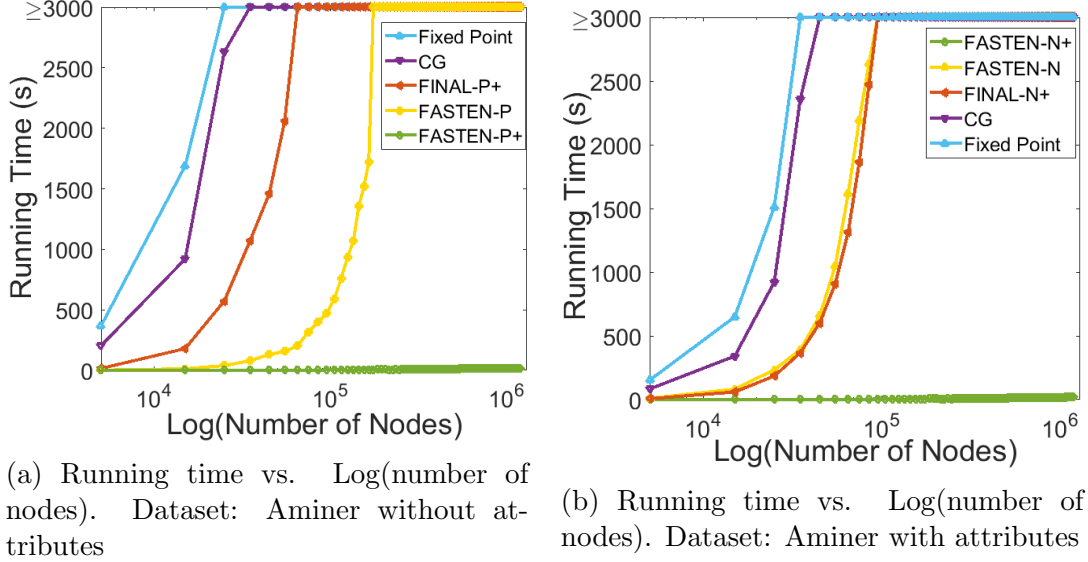


Figure 3.4: Scalability on plain (left) and attributed graphs (right). Best viewed in color.

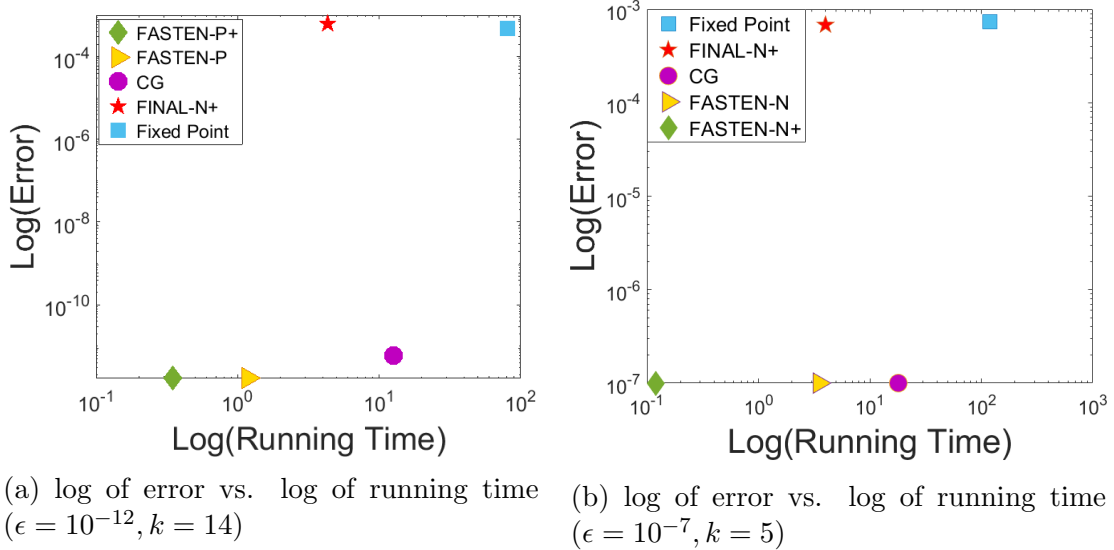


Figure 3.5: Error vs. running time comparison of five methods on plain graphs (left) and attributed graphs (right). Best viewed in color.

Effectiveness. For the effectiveness evaluation, we define the error of the solution as: $Error = \|\mathbf{X} - \mathbf{X}'\|_F$ where \mathbf{X} is the solution matrix by FASTEN or other baseline methods, and \mathbf{X}' is computed by the direct method on the equivalent linear system of the Sylvester equation (namely Equation (3.2) and (3.3)). Since the direct method takes $O(n^6)$ in time, we use a subset of *AMiner* with $4K$ nodes in this experiment to avoid extremely long running time of the direct method. We compare the *Error* vs. the running time of our methods with

all the baseline methods in Figure 3.5. We can observe that the *Conjugate Gradient (CG)* method and all of our developed FASTEN algorithms have a very small *Error* (less than 10^{-7}). *Error* of both *Fixed Point* and *FINAL-N+* are more than 10^{-4} . In the meanwhile, the running time of the developed FASTEN is smaller than all baseline methods in all cases.

Parameter Sensitivity.

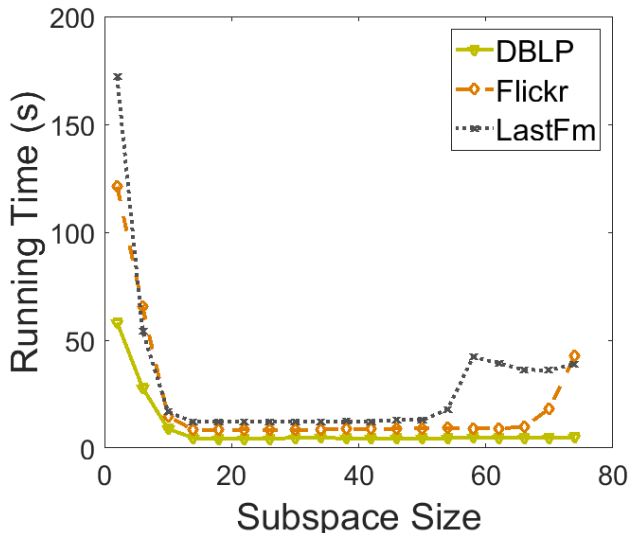


Figure 3.6: Sensitivity study of FASTEN-P. Best viewed in color.

In the developed FASTEN algorithms, we need to set Krylov subspace size k , which might affect the convergence rate and the running time of the developed algorithms. Generally speaking, a smaller k makes the computation of the inner loop faster, but might cause a slower convergence of the outer loop (see Section 3.1.2 and Section 3.2 for the detailed descriptions of inner loop and outer loop); on the other hand, it would take longer time for the inner-loop with a larger k although it might help reduce the iteration number of the outer-loop.

Take FASTEN-P as an example, we report the running time of FASTEN-P vs. the subspace size k on three datasets in Figure 3.6. We can see the running time stays stable at a low number when $14 \leq k \leq 60$, and it starts to increase when k is outside this range. Overall, we found that the running time of all the four developed algorithms is insensitive in a relatively large range of the Krylov subspace size k .

3.3 NOVEL APPLICATION: INTERACTIVE SUBGRAPH MATCHING

Many real networks often accompany with rich node and/or edge attribute, including the demographic information for users on a social network (i.e., node attribute), the transac-

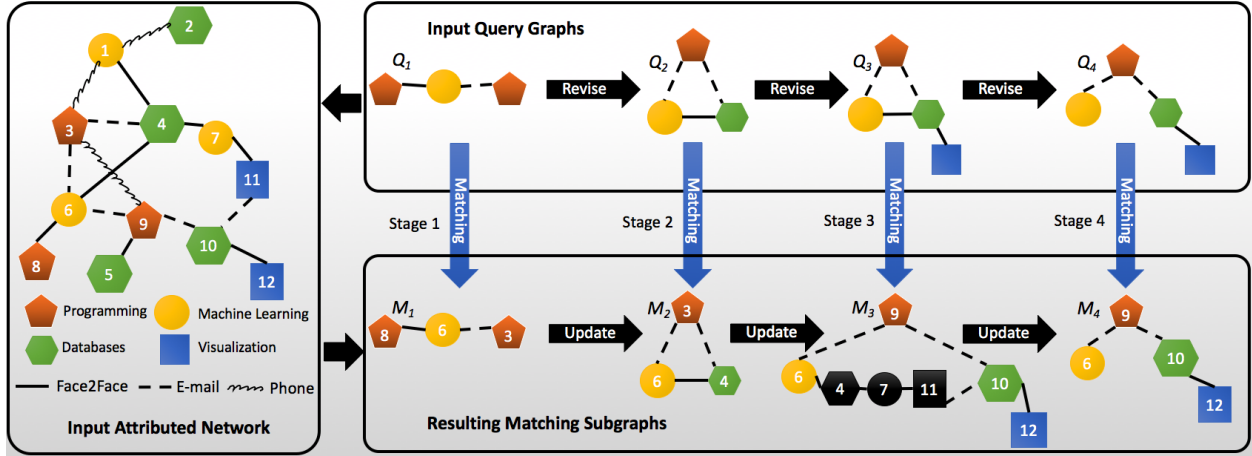


Figure 3.7: An illustrative example of interactive attributed subgraph matching. Best viewed in color.

tion types on a financial transaction network (i.e., edge attribute), the expertise of team members as well as the communication channels between them on a collaboration network (i.e., both node and edge attributes). Attribute subgraph matching [105, 130] is the key for many explorative mining tasks, i.e., to help identify *user-specific* patterns from such attributed networks, and has become an integral part of some emergent visual graph analytic platforms [131]. To name a few, in network science of teams, attributed subgraph matching is the cornerstone to help form a team of experts with desired skills of each member as well as the communication pattern between team members (i.e., example-based team formation) [132, 133]; in finance informatics, it is a powerful tool to identify suspicious transaction patterns (e.g., money laundry ring) [105]; in intelligence analysis and law enforcement, it can help end-analyst generate valuable leads (e.g., a suspicious terror plot, the master-criminal mind, etc.) [108].

Despite that tremendous progress has been made, most, if not all, of the existing attributed subgraph matching algorithms requires the user accurately knows what s/he is looking for, in other words, to provide an accurate query graph. However, in some application scenarios, the end-user might only have a vague idea on her search intent at the beginning and thus needs to constantly revise and refine her initial query graph.

Figure 3.7 represents an illustrative example of such interactive matching process. On the left side of Figure 3.7 is the input data (attributed) data network, and on the right side we illustrate a procedure of team formation in an interactive style. Let us assume skill A to skill D stands for *programming*, *databases*, *machine learning* and *visualization*, respectively, and the edge attribute 1 to 3 represents three different communication approaches. We have the

following interactive matching process. (1) At the beginning, the user only knows s/he wants to form a team of size 3, with two skills (e.g., programming and machine learning) and the team should be led by the machine learning expert, and therefore s/he issues a line query (Q_1). (2) After the user sees the initial matching graph (M_1), s/he realizes that the project only needs one programmer; but in the meanwhile it requires another expert in *databases* and better communication between all the team members. Therefore, s/he issues a revised clique query (Q_2). (3) After seeing the corresponding updated matching subgraph (M_2), s/he decides to expand the team size by adding an additional expert in data visualization to help databases expert. Thus, s/he expands the previous query graph by adding an additional link and node (Q_3). (4) After s/he sees the updated matching result (M_3), s/he finds that having too many communications (i.e., over-communications) between the team members might hurt the team productivity. Thus, s/he revises the query graph again (Q_4), to keep only vital communications between the key team members. Finally the user finds the ideal team (M_4).

In such an interactive setting, a major bottleneck is the computational efficiency. This is because simply re-running the matching algorithms on the revised query graph from scratch might be computationally too costly as such algorithms require either building an index of the underlying data graph (e.g., [109]) or a costly iterative process during the query stage (e.g., [105, 130]).

To address these limitations, we develop a family of effective and efficient algorithms (FIRST) to support interactive attributed subgraph matching scenario. There are two key ideas behind the developed methods. The first is to recast the attributed subgraph matching problem as a cross-network node similarity problem. This formulation allows us to simultaneously encode topology consistency and attribute consistency in a coherent optimization problem, whose major computation lies in solving a Sylvester equation for the query and the underlying data graph [30]. The second key idea is to explore the smoothness between the initial and revised queries, which means that the revised query can often be viewed as a perturbed version of the previous query graph. For the example in Figure 3.7, the third query graph (Q_3) can be viewed as perturbed version of the second query (Q_2) with an additional node with one additional link. It turns out this observation enable us to solve the new/updated Sylvester equation incrementally, without re-solving it from scratch. The developed FIRST algorithms enjoy a *linear* time complexity with respect to the input data network size. We conduct extensive experiments on real-world datasets, which show that the developed method leads to up to $16\times$ speed-up with more than 90% accuracy.

3.3.1 Problem Definition

Table 3.4 summarizes the main symbols and notation used throughout the work. We use bold uppercase letters for matrices (e.g., A), bold lowercase letters for vectors (e.g., s), and lowercase letters (e.g., α) for scalars. We use the calligraphic letter \mathcal{G} to represent an attributed network, i.e., $\mathcal{G} = (\mathbf{A}, \mathbf{N}_{\mathbf{A}}, \mathbf{E}_{\mathbf{A}})$, where \mathbf{A} is the adjacency matrix, $\mathbf{N}_{\mathbf{A}}$ and $\mathbf{E}_{\mathbf{A}}$ are the node and edge attribute matrices of \mathcal{G} , respectively. We use the subscript q to denote the corresponding notations for the query graph (i.e., $\mathcal{Q} = (\mathbf{A}_q, \mathbf{N}_q, \mathbf{E}_q)$), and $\tilde{\cdot}$ to denote the corresponding notation after the user modifies the initial query (i.e., $\tilde{\mathcal{Q}} = (\tilde{\mathbf{A}}_q, \tilde{\mathbf{N}}_q, \tilde{\mathbf{E}}_q)$ is the revised query graph). Likewise, the initial and updated similarity matrix are denoted by \mathbf{s} and $\tilde{\mathbf{s}}$. The initial matching subgraph and the updated matching subgraph are denoted by \mathcal{M} and $\tilde{\mathcal{M}}$ respectively. Additionally, We use $\hat{\cdot}$ to denote the approximate version of vectors or matrices in this work (e.g., $\hat{\mathbf{s}}$, $\hat{\mathbf{W}}^{sym}$, etc.).

The node attribute matrix of input networks \mathbf{N} is defined as $\mathbf{N} = \sum_{p=1}^K \mathbf{N}_{\mathbf{A}}^p \otimes \mathbf{N}_{\mathbf{q}}^p$ where K is the number of distinct node labels. $\mathbf{N}_{\mathbf{A}}^p$ and $\mathbf{N}_{\mathbf{q}}^p$ are diagonal matrices in which $\mathbf{N}_{\mathbf{A}}^p(a, a) = 1$ if the node a in network \mathcal{G} has node attribute k and otherwise it is equal to 0. The edge attribute matrix of input networks \mathbf{E} is defined as $\mathbf{E} = \sum_{l=1}^L \mathbf{E}_{\mathbf{A}}^l \otimes \mathbf{E}_{\mathbf{q}}^l$ where L is the number of distinct edge labels. $\mathbf{E}_{\mathbf{A}}^l$ and $\mathbf{E}_{\mathbf{q}}^l$ are $n \times n$ and $k \times k$ matrices respectively. $\mathbf{E}_{\mathbf{A}}^l(a, b) = 1$ if the edge (a, b) in network \mathcal{G} has edge attribute l and otherwise it is equal to 0. For example, for the initial query graph (\mathcal{Q}_1) in Figure 3.7, we have $\mathbf{N}_{\mathbf{q}}^1(1, 1) = 1$, $\mathbf{N}_{\mathbf{q}}^2(1, 1) = 0$, $\mathbf{E}_{\mathbf{q}}^1(1, 2) = 1$ and $\mathbf{E}_{\mathbf{q}}^2(1, 2) = 0$, etc. For the revised query graph, we have $\mathbf{E}_{\mathbf{q}}^1(2, 3) = 1$ and $\mathbf{N}_{\mathbf{q}}^3(3, 3) = 1$, etc.

With these notations, the interactive attributed subgraph matching problem can be formally defined as:

Problem 3.1. INTERACTIVE ATTRIBUTE SUBGRAPH MATCHING.

Given: (1) an undirected attributed network \mathcal{G} , (2) an undirected initial query graph \mathcal{Q} , (3) the initial matching subgraph \mathcal{M} , (4) the revised query graph $\tilde{\mathcal{Q}}$;

Output: the updated matching subgraph $\tilde{\mathcal{M}}$.

For the team formation example in Figure 3.7, between stage-1 and stage-2, the line query (\mathcal{Q}_1) and the clique query (\mathcal{Q}_2) are the initial and the revised query graphs, respectively; and M_1 and M_2 are the corresponding initial and revised matching subgraph, respectively. Between stage-2 and stage-3, the clique query (\mathcal{Q}_2) will be treated as the initial query graph, and \mathcal{Q}_3 becomes the new revised query graph, so on and so forth.

Table 3.4: Symbols and Definition

Symbols	Definition
$\mathcal{G} = \{\mathbf{A}, \mathbf{N}, \mathbf{E}\}$	an attributed network
$\mathcal{Q}, \tilde{\mathcal{Q}}$	initial and revised query network
$\mathcal{M}, \tilde{\mathcal{M}}$	initial and updated matching subgraph
\mathbf{A}, \mathbf{A}_q	the adjacency matrix of the attributed data network and query graph
$\mathbf{N}_A/\mathbf{N}_q, \mathbf{E}_A/\mathbf{E}_q$	the node and edge attribute matrix of the network/query graph
n, k	# of nodes in \mathcal{G} and \mathcal{Q}
m_1, m_2	# of edges in \mathcal{G} and \mathcal{Q}
P, L	# of the node and edge labels
a, b	node/edge indices of \mathcal{G}
x, y	node/edge indices of \mathcal{Q}
p, l	node/edge label indices
\mathbf{I}	an identity matrix
\mathbf{H}	$k \times n$ prior alignment preference
\mathbf{S}	$k \times n$ similarity matrix
r, t	reduced ranks
α	the parameter, $0 < \alpha < 1$
$s = \text{vec}(S)$	vectorize a matrix S in column order
$Q = \text{mat}(q, n_2, n_1)$	reshape vector q into an $n_2 \times n_1$ matrix in column order
\mathbf{W}^{sym}	symmetrically normalize matrix W
$\mathbf{D} = \text{diag}(\mathbf{d})$	diagonalize a vector \mathbf{d}
\otimes	Kronecker product
\odot	element-wise matrix product
$\text{abs}()$	absolute value
$\ \cdot\ _F$	Frobenius norm
$\text{and}()$	AND operation

3.3.2 Fast Interactive Algorithms

In this section, we first review an existing network alignment algorithm, which provides the base for our developed algorithm. Then, we present our algorithms (FIRST) in different scenarios, e.g., revising the topology/node attributes/edge attributes in the query graph, whether the input data network has both node and edge attributes, etc.

Preliminaries. Generally speaking, in attributed subgraph matching, we want to find a subgraph from the input data network \mathcal{G} that maximizes some "goodness" function with regard to the query graph \mathcal{Q} [105]. Here, our idea is to recast it as a cross-network node similarity problem. To be specific, let \mathbf{S} be a $k \times n$ non-negative cross-network node-similarity matrix, where $\mathbf{S}(i, j)$ measures the cross-network similarity between the i^{th} query node in \mathcal{Q} and the j^{th} node in \mathcal{G} (i.e., to what extent the j^{th} node in \mathcal{G} matches the i^{th} query node).

In order to find the cross-network node similarity matrix \mathbf{S} , we adopt a recent network alignment algorithm [30], which naturally encodes both the topological and attribute consistency between two networks (the data network and the query graph in our setting) in the

following Sylvester equation (please refer to [30] for the full details).

$$\mathbf{s} = \alpha \mathbf{W}^{sym} \mathbf{s} + (1 - \alpha) \mathbf{h} \quad (3.20)$$

where $\mathbf{s} = \text{vec}(\mathbf{S})$, $\mathbf{h} = \text{vec}(\mathbf{H})$ and \mathbf{H} is a $k \times n$ matrix of prior similarity knowledge. $\mathbf{W}^{sym} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$ is the symmetrical normalization of \mathbf{W} and \mathbf{W} can take three possible forms according to the availability of the attribute information in the networks [30], including

- (i) $\mathbf{W}^{sym} = \mathbf{A} \otimes \mathbf{A}_q$: if only adjacency matrix is available;
- (ii) $\mathbf{W}^{sym} = \mathbf{N}(\mathbf{A} \otimes \mathbf{A}_q)\mathbf{N}$: if the adjacency matrix and node attributes are available but edge attributes are missing;
- (iii) $\mathbf{W}^{sym} = \mathbf{N}[\mathbf{E} \odot (\mathbf{A} \otimes \mathbf{A}_q)]\mathbf{N}$: if the adjacency matrix, node and edge attributes are all available.

And \mathbf{D} is the diagonal degree matrix of \mathbf{W} . For example, if both node and edge attributes are available (\mathbf{W}^{sym} being type (iii)), \mathbf{D} is computed by:

$$\mathbf{D} = \text{diag}\left(\sum_{k,k'=1}^K \sum_{l=1}^L (\mathbf{N}_A^k (\mathbf{E}_A^l \odot \mathbf{A}) \mathbf{N}_A^{k'} \mathbf{1}) \otimes (\mathbf{N}_q^k (\mathbf{E}_q^l \odot \mathbf{A}_q) \mathbf{N}_q^{k'} \mathbf{1})\right) \quad (3.21)$$

where K and L are the number of different node and edge labels respectively.

The solution of Eq. (3.20) can be obtained by either an iterative procedure or a closed-form formula. And the closed-form solution can be further approximated (and sped up) by using low-rank approximation on the two input networks. However, none of these solutions is applicable in the interactive setting. This is because: (1) for the iterative solution, each iteration requires $O(\min(km_1, nm_2))$ time complexity and it might take many iterations to compute the similarity matrix \mathbf{S} only for the initial query network, let alone for the interactively updated query networks, and (2) for the closed-form solution, its $O(n^3 k^3)$ time complexity, or even its approximate solution, is still computationally too costly if user frequently changes the queries.

The common key ideas behind our upcoming developed algorithm FIRST are that: (1) we recast the attributed subgraph matching problem as a cross-network node similarity problem (i.e., to compute the matrix \mathbf{S} in Eq. (3.20)), and (2) by exploring the smoothness between the initial query and updated queries, we can solve the Sylvester equation incrementally. In our work, we assume that the data network \mathcal{G} and the prior preference \mathbf{H} remain unchanged during the interactive process. In practice, the size of the query network is often much smaller than that of the data network, i.e. $k \ll n$.

Handling Node Attribute. In this subsection we consider the scenario in which node attribute is available but edge attribute is missing in both network and query graph. We discuss two cases: (A) only revising the topology of the query graph and (B) only revising node attribute of the query graph. First we present Algorithm 3.7 to solve the case where only topology is changed.

Topology Change. Based on our problem formulation and assumptions, in the interactive scenario the updated similarity vector $\tilde{\mathbf{s}}$ after query modification can be expressed as follows:

$$\tilde{\mathbf{s}} = (1 - \alpha)(\mathbf{I} - \alpha\tilde{\mathbf{W}}^{sym})^{-1}\mathbf{h} \quad (3.22)$$

where $\tilde{\mathbf{W}}^{sym} = \mathbf{D}^{-1/2}\mathbf{N}(\mathbf{A} \otimes \mathbf{A}_q)\mathbf{N}\mathbf{D}^{-1/2}$. Since only the topology of the updated query differs from initial query, the node attribute information \mathbf{N} will not contribute to $\tilde{\mathbf{s}}$.

Since many real-world networks are observed to have a low-rank structure, we can leverage this characteristics to obtain a good approximation of the similarity matrix. Here, we define the approximate similarity matrix as follows:

Definition 3.1. (APPROXIMATE SIMILARITY VECTOR)

Given a similarity vector $\mathbf{s} = (1 - \alpha)(\mathbf{I} - \alpha\mathbf{W}^{sym})^{-1}\mathbf{h}$, its approximate similarity vector $\hat{\mathbf{s}}$ is given by

$$\hat{\mathbf{s}} = (1 - \alpha)(\mathbf{I} - \alpha\hat{\mathbf{W}}^{sym})^{-1}\mathbf{h} \quad (3.23)$$

where $\hat{\mathbf{W}}^{sym}$ is a low rank approximation of \mathbf{W}^{sym} .

Since the adjacency matrices \mathbf{A} and \mathbf{A}_q are both symmetric, we can apply rank- r eigenvalue decomposition (EVD) on \mathbf{A} and \mathbf{A}_q . An additional advantage of using EVD is that we can reduce the space complexity by only storing the low-rank matrices instead of the whole adjacency matrices. The algorithm is summarized in Algorithm 3.7.

From the fourth line to seventh line are the precomputing stage. The top r eigenvalue decomposition of \mathbf{A} and the top t eigenvalue decomposition of \mathbf{A}_q are calculated. $\mathbf{U}_\mathbf{A}$ and $\mathbf{\Lambda}_\mathbf{A}$ are stored. In the interactive stage, only the top t eigenvalue decomposition of $\tilde{\mathbf{A}}_q$ is calculated. Then $\tilde{\mathbf{S}}$ is computed from line 10 to line 15. The proof of correctness of FIRST-Q is presented as follows:

Theorem 3.3 (Correctness of FIRST-Q). The Algorithm 3.7 (FIRST-Q) gives the approximate similarity vector by Definition 3.1: $\tilde{\mathbf{s}} = \hat{\mathbf{s}}$.

Proof. According to Definition 3.1,

$$\begin{aligned} \hat{\mathbf{s}} &= (1 - \alpha)[\mathbf{I} - \alpha\mathbf{P}_1(\hat{\mathbf{A}} \otimes \hat{\mathbf{A}}_q)\mathbf{P}_1]^{-1}\mathbf{h} \\ &= (1 - \alpha)\mathbf{P}_1^{-1}[\mathbf{D}_1 - \alpha(\hat{\mathbf{A}} \otimes \hat{\mathbf{A}}_q)]^{-1}\mathbf{P}_1^{-1}\mathbf{h} \end{aligned} \quad (3.24)$$

Algorithm 3.7 FIRST-Q

Input: the attributed data network $\mathcal{G} = \{\mathbf{A}, \mathbf{N}_A\}$,

1: the initial and revised query network $\mathcal{Q} = \{\mathbf{A}_q, \mathbf{N}_q\}$, $\tilde{\mathcal{Q}} = \{\tilde{\mathbf{A}}_q, \tilde{\mathbf{N}}_q\}$,

2: the alignment preference matrix \mathbf{H} ,

3: parameter α .

Output: Approx. updated similarity matrix $\tilde{\mathbf{S}}$.

4: *Precomputing Stage:*

5: $\mathbf{U}_A \Lambda_A \mathbf{U}_A^T \leftarrow \mathbf{A}$; //top r eigenvalue decomposition

6: $\mathbf{U}_Q \Lambda_Q \mathbf{U}_Q^T \leftarrow \mathbf{A}_q$; //top t eigenvalue decomposition

7: Store \mathbf{U}_A, Λ_A ;

8: *Interactive Stage:*

9: $\mathbf{U}_Q \Lambda_Q \mathbf{U}_Q^T \leftarrow \tilde{\mathbf{A}}_q$; //top t eigenvalue decomposition

10: Compute node attribute matrix of input networks \mathbf{N} and diagonal degree matrix \mathbf{D} ; Construct $\mathbf{P} = \mathbf{D}^{-\frac{1}{2}} \mathbf{N}$, $\mathbf{D}_1 = \mathbf{P}^{-1} \mathbf{P}^{-1}$;

11: $\mathbf{L} \leftarrow \mathbf{U}_A \otimes \mathbf{U}_Q$;

12: $\mathbf{R} \leftarrow \mathbf{U}_A^T \otimes \mathbf{U}_Q^T$;

13: $\Lambda \leftarrow \Lambda_A \otimes \Lambda_Q$;

14: $\tilde{\mathbf{s}} = (1 - \alpha) \mathbf{P}^{-1} [\mathbf{D}_1^{-1} + \alpha \mathbf{D}_1^{-1} \mathbf{L} (\Lambda^{-1} - \alpha \mathbf{R} \mathbf{D}_1^{-1} \mathbf{L})^{-1} \mathbf{R} \mathbf{D}_1^{-1}] \mathbf{P}^{-1} \mathbf{h}$;

15: $\tilde{\mathbf{S}} = \text{mat}(\tilde{\mathbf{s}}, n, k)$; //reshape similarity vector

in which $\mathbf{P}_1 = \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{N} = \mathbf{N} \tilde{\mathbf{D}}^{-\frac{1}{2}}$, $\mathbf{D}_1 = \mathbf{P}_1^{-1} \mathbf{P}_1^{-1}$. Let $\mathbf{U}_A \Lambda_A \mathbf{U}_A^T$ be top r eigenvalue decomposition of \mathbf{A} and $\mathbf{U}_Q \Lambda_Q \mathbf{U}_Q^T$ be top t eigenvalue decomposition of \mathbf{A}_q . Then in the interactive stage, $\hat{\mathbf{W}}^{sym}$ can be written as:

$$\begin{aligned} \hat{\mathbf{W}}^{sym} &= \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{N} [(\mathbf{U}_A \Lambda_A \mathbf{U}_A^T) \otimes (\mathbf{U}_Q \Lambda_Q \mathbf{U}_Q^T)] \mathbf{N} \tilde{\mathbf{D}}^{-\frac{1}{2}} \\ &= \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{N} [(\mathbf{U}_A \otimes \mathbf{U}_Q) (\Lambda_A \otimes \Lambda_Q) (\mathbf{U}_A^T \otimes \mathbf{U}_Q^T)] \mathbf{N} \tilde{\mathbf{D}}^{-\frac{1}{2}} \end{aligned} \quad (3.25)$$

Let $\mathbf{L} = \mathbf{U}_A \otimes \mathbf{U}_Q$, $\mathbf{R} = \mathbf{U}_A^T \otimes \mathbf{U}_Q^T$, $\Lambda = \Lambda_A \otimes \Lambda_Q$. According to equation 3.24 and the definition of $\hat{\mathbf{s}}$,

$$\begin{aligned} \hat{\mathbf{s}} &= (1 - \alpha) (\mathbf{I} - \alpha \mathbf{P}_1 \mathbf{L} \Lambda \mathbf{R} \mathbf{P}_1)^{-1} \mathbf{h} \\ &= (1 - \alpha) \mathbf{P}_1^{-1} (\mathbf{D}_1 - \alpha \mathbf{L} \Lambda \mathbf{R})^{-1} \mathbf{P}_1^{-1} \mathbf{h} \\ &= (1 - \alpha) \mathbf{P}_1^{-1} [\mathbf{D}_1^{-1} + \alpha \mathbf{D}_1^{-1} \mathbf{L} (\Lambda^{-1} - \alpha \mathbf{R} \mathbf{D}_1^{-1} \mathbf{L})^{-1} \mathbf{R} \mathbf{D}_1^{-1}] \mathbf{P}_1^{-1} \mathbf{h} \end{aligned} \quad (3.26)$$

where the third equality comes from the Sherman-Morrison Lemma [134]. Hence the correctness of FIRST-Q is proved. QED.

It is worth pointing out that if the node attribute information is also missing, which means that \mathbf{W}^{sym} takes the form of type (i), the algorithm also works by setting \mathbf{N} to be identity matrix \mathbf{I} . The proof is almost identical to the proof above.

Node Attribute Change. Here, we provide an algorithm (FIRST-N) for the scenario where

only node attribute of the query graph is revised during user’s interactive query process. Again, the edge attribute is not available in this scenario (i.e., no \mathbf{E} and \mathbf{E}_q). The algorithm is summarized in Algorithm 3.8.

Algorithm 3.8 FIRST-N

Input: the attributed data network at time step 1 $\mathcal{G} = \{\mathbf{A}, \mathbf{N}_A\}$,
1: the initial and revised query network $\mathcal{Q} = \{\mathbf{A}_q, \mathbf{N}_q\}$, $\tilde{\mathcal{Q}} = \{\tilde{\mathbf{A}}_q, \tilde{\mathbf{N}}_q\}$,
2: the alignment preference matrix \mathbf{H} ,
3: parameter α .
Output: Approx. updated similarity matrix $\tilde{\mathbf{S}}$.
4: *Precomputing Stage:*
5: $\mathbf{U}_A \Lambda_A \mathbf{U}_A^T \leftarrow \mathbf{A}$; //top r eigenvalue decomposition;
6: $\mathbf{U}_Q \Lambda_Q \mathbf{U}_Q^T \leftarrow \mathbf{A}_q$; //top t eigenvalue decomposition;
7: $\mathbf{L} \leftarrow \mathbf{U}_A \otimes \mathbf{U}_Q$;
8: $\mathbf{R} \leftarrow \mathbf{U}_A^T \otimes \mathbf{U}_Q^T$;
9: $\Lambda \leftarrow \Lambda_A \otimes \Lambda_Q$;
10: Store \mathbf{L} , \mathbf{R} , Λ ;
11: *Interactive Stage:*
12: Compute node attribute matrix of input matrix \mathbf{N} and diagonal degree matrix \mathbf{D} with $\tilde{\mathbf{N}}_q$, \mathbf{A} and $\tilde{\mathbf{A}}_q$; Compute $\mathbf{P} = \mathbf{D}^{-\frac{1}{2}} \tilde{\mathbf{N}}_q$;
13: Compute $\mathbf{D}_1 = \mathbf{P}^{-1} \mathbf{P}^{-1}$;
14: $\tilde{\mathbf{s}} = (1 - \alpha) \mathbf{P}^{-1} [\mathbf{D}_1^{-1} + \alpha \mathbf{D}_1^{-1} \mathbf{L} (\Lambda^{-1} - \alpha \mathbf{R} \mathbf{D}_1^{-1} \mathbf{L})^{-1} \mathbf{R} \mathbf{D}_1^{-1}] \mathbf{P}^{-1} \mathbf{h}$;
15: $\tilde{\mathbf{S}} = \text{mat}(\tilde{\mathbf{s}}, n, k)$; //reshape similarity vector

In the precomputing stage, the algorithm calculates and stores \mathbf{L} , \mathbf{R} and Λ , while in the interactive stage the algorithm can directly construct \mathbf{P} . $\tilde{\mathbf{S}}$ is calculated from line 12 to line 15.

From the algorithm, we can notice that the eigenvalue decomposition of adjacency matrix \mathbf{A} and \mathbf{A}_q , the construction of matrices \mathbf{L} , \mathbf{R} and Λ can be precomputed at initial step, thus further speed up this algorithm, compared with Algorithm 3.7. The proof of the correctness of Algorithm 3.8 is similar to the proof of Algorithm 3.7 and is omitted for space.

Handling Edge Attribute. In this subsection, we discuss the interactive scenario where both node and edge attribute are available. Note that in this case the query can be revised in multiple ways (e.g., only revising network topology/node attribute/edge attribute vs simultaneously revising network topology as well as attributes, etc.). In our developed algorithm (FIRST-E), we consider the general case where network topology, node and edge attributes are all revised simultaneously during one interactive stage. The rest of the ways to revise the query graph are special cases, and thus can also be supported by our approach.

The algorithm is presented in Algorithm 3.9.

The precomputing stage is from line 4 to line 14 and the interactive stage is from line 16 to line 24. In line 20, two block matrices (\mathbf{U} , Λ) are constructed. \mathbf{U} is a $1 \times L$ block matrix

Algorithm 3.9 FIRST-E

Input: the attributed network at time step 1 $\mathcal{G} = \{\mathbf{A}, \mathbf{N}_A, \mathbf{E}_A\}$,
 1: the initial and revised query network $\mathcal{Q} = \{\mathbf{A}_q, \mathbf{N}_q, \mathbf{E}_q\}$, $\tilde{\mathcal{Q}} = \{\tilde{\mathbf{A}}_q, \tilde{\mathbf{N}}_q, \tilde{\mathbf{E}}_q\}$
 2: the alignment preference matrix \mathbf{H} ,
 3: parameter α , index of changed edge attribute I' (optional).
Output: Approx. updated similarity matrix $\tilde{\mathbf{S}}$.

- 4: **Precomputing Stage:**
- 5: **for** each $l \in [1, L]$ **do**
- 6: $\mathbf{U}_A^l \Lambda_A^l (\mathbf{U}_A^l)^T \leftarrow \mathbf{E}_A^l \odot \mathbf{A}$; //top r eigenvalue decomposition
- 7: Store $\mathbf{U}_A^l, \Lambda_A^l$;
- 8: **end for**
- 9: **if** I' is not empty **then**
- 10: **for** each $k \in [1, L]$ **do**
- 11: $\mathbf{U}_q^k \Lambda_q^k (\mathbf{U}_q^k)^T \leftarrow \mathbf{E}_q^k \odot \mathbf{A}_q$; //top t eigenvalue decomposition
- 12: Store $\mathbf{U}_q^k, \Lambda_q^k$;
- 13: **end for**
- 14: **end if**
- 15: **Interactive Stage:**
- 16: Construct \mathbf{N} and \mathbf{D} from $\mathbf{A}, \tilde{\mathbf{A}}_q, \mathbf{N}_A, \tilde{\mathbf{N}}_q, \mathbf{E}_A, \tilde{\mathbf{E}}_q$;
- 17: **for** each $l \in [1, L]$ (or each $l \in I'$ if I' is not empty) **do**
- 18: $\mathbf{U}_q^l \Lambda_q^l (\mathbf{U}_q^l)^T \leftarrow \mathbf{E}_q^l \odot \tilde{\mathbf{A}}_q$; //top t eigenvalue decomposition
- 19: **end for**
- 20: Construct block matrix $\mathbf{U} = [\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_L]$, in which $\mathbf{V}_i = \mathbf{U}_A^i \otimes \mathbf{U}_q^i$ ($i \in [1, L]$), and block matrix $\mathbf{\Lambda} = \text{diag}(\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_L)$, in which $\mathbf{Y}_j = \Lambda_A^j \otimes \Lambda_q^j$ ($j \in [1, L]$);
- 21: $\mathbf{L} \leftarrow \mathbf{D}^{-\frac{1}{2}} \mathbf{N} \mathbf{U}$;
- 22: $\mathbf{R} \leftarrow \mathbf{U}^T \mathbf{N} \mathbf{D}^{-\frac{1}{2}}$;
- 23: $\tilde{\mathbf{s}} = (1 - \alpha)[\mathbf{I} + \alpha \mathbf{L}(\mathbf{\Lambda}^{-1} - \alpha \mathbf{R} \mathbf{L})^{-1} \mathbf{R}] \mathbf{h}$;
- 24: $\tilde{\mathbf{S}} = \text{mat}(\tilde{\mathbf{s}}, n, k)$; //reshape similarity vector

with each element being \mathbf{V}_i , while $\mathbf{\Lambda}$ is a $L \times L$ diagonal block matrix with each diagonal element being \mathbf{Y}_j ($i, j \in [1, L]$).

As mentioned at the beginning of this subsection, the algorithm still works in other ways to revise the query graph by setting $\tilde{\mathbf{A}}_q, \tilde{\mathbf{N}}_q$ or $\tilde{\mathbf{E}}_q$ equal to their initial counterparts. Specifically, if only certain edge attributes are changed, the algorithm could take an optional parameter I' (line 3) as the index of changed edge attribute, otherwise I' is set empty. In line 9, the top t eigenvalue decomposition of the element-wise product of the k^{th} edge attribute matrix \mathbf{E}_q^k and network adjacency matrix \mathbf{A} is computed, if I' is not empty. In the following interactive stage, the eigenvalue decomposition can be only calculated on the element-wise product of the changed edge attribute matrices (\mathbf{E}_q^l) and $\tilde{\mathbf{A}}_q$ (line 17 to 19), which further speed up the interactive computing stage.

Theorem 3.4 (Correctness of *FIRST-E*). The Algorithm 3.9 (*FIRST-E*) gives the approx-

imate updated similarity vector by Definition 3.1: $\tilde{\mathbf{s}} = \hat{\mathbf{s}}$.

Proof. We know that $\tilde{\mathbf{W}}$ takes the form of type C (which is $\mathbf{D}^{-1/2}\mathbf{N}[\mathbf{E} \odot (\mathbf{A} \otimes \mathbf{A}_q)]\mathbf{ND}^{-1/2}$). Then

$$\begin{aligned}\tilde{\mathbf{W}}^{\text{sym}} &= \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\left[\left(\sum_{l=1}^L \mathbf{E}_A^l \otimes \mathbf{E}_q^l\right) \odot (\mathbf{A} \otimes \tilde{\mathbf{A}}_q)\right]\mathbf{ND}^{-\frac{1}{2}} \\ &= \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\left[\sum_{l=1}^L (\mathbf{E}_A^l \odot \mathbf{A}) \otimes (\mathbf{E}_q^l \odot \tilde{\mathbf{A}}_q)\right]\mathbf{ND}^{-\frac{1}{2}}\end{aligned}\tag{3.27}$$

$$\begin{aligned}\hat{\mathbf{W}}^{\text{sym}} &= \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\left[\sum_{l=1}^L (\mathbf{U}_A^l \Lambda_A^1 (\mathbf{U}_A^l)^T) \otimes (\mathbf{U}_q^l \Lambda_q^1 (\mathbf{U}_q^l)^T)\right]\mathbf{ND}^{-\frac{1}{2}} \\ &= \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\left[\sum_{l=1}^L (\mathbf{U}_A^l \otimes \mathbf{U}_q^l) (\Lambda_A^1 \otimes \Lambda_q^1) ((\mathbf{U}_A^l)^T \otimes (\mathbf{U}_q^l)^T)\right]\mathbf{ND}^{-\frac{1}{2}} \\ &= \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\mathbf{U}\Lambda\mathbf{U}^T\mathbf{ND}^{-\frac{1}{2}}\end{aligned}\tag{3.28}$$

where \mathbf{U} and Λ are block matrices as described in line 20 of Algorithm 3.9. From Equation (3.27) to Equation (3.28), the top r and top t eigenvalue decomposition are taken as described in line 6 and line 18. The derivation from the second line to the third line in Equation (3.28) is based on the property of block matrix [135]. If I' is not empty, in the interactive stage,

$$\begin{aligned}\hat{\mathbf{W}}^{\text{sym}} &= \mathbf{D}^{-\frac{1}{2}}\mathbf{N}\left[\sum_{l \in I'} (\mathbf{U}_A^l \Lambda_A^1 (\mathbf{U}_A^l)^T) \otimes (\mathbf{U}_q^l \Lambda_q^1 (\mathbf{U}_q^l)^T)\right] \\ &\quad + \sum_{l \notin I'} (\mathbf{U}_A^l \Lambda_A^1 (\mathbf{U}_A^l)^T) \otimes (\mathbf{U}_q^l \Lambda_q^1 (\mathbf{U}_q^l)^T)\mathbf{ND}^{-\frac{1}{2}}\end{aligned}\tag{3.29}$$

where the eigen-decomposed term that is not in the index I' (in the second line of equation 9) is calculated in the precomputing stage (line 6). Still, equation 9 is equal to equation 8.

Let $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{N}\mathbf{U}$, $\mathbf{R} = \mathbf{U}^T\mathbf{N}\mathbf{D}^{-1/2}$. Then $\hat{\mathbf{s}}$ is given as:

$$\begin{aligned}\hat{\mathbf{s}} &= (1 - \alpha)(\mathbf{I} - \alpha\hat{\mathbf{W}})^{-1}\mathbf{h} \\ &= (1 - \alpha)(\mathbf{I} - \alpha\mathbf{L}\mathbf{\Lambda}\mathbf{R})^{-1}\mathbf{h} \\ &= (1 - \alpha)[\mathbf{I} + \alpha\mathbf{L}(\mathbf{\Lambda}^{-1} - \alpha\mathbf{R}\mathbf{L})^{-1}\mathbf{R}]\mathbf{h}\end{aligned}$$

The last line comes from Sherman-Morrison Lemma [134]. Hence we have proved that $\tilde{\mathbf{s}} = \hat{\mathbf{s}}$ and FIRST-E gives the approximate updated similarity vector. QED.

Implementation Details. In this section, we present implementation details of our method to transform the similarity matrix to one or more matching subgraphs. After either FIRST-Q, FIRST-N or FIRST-E is called, the returned similarity matrix $\tilde{\mathbf{S}}$ should be transferred into updated matching subgraph $\tilde{\mathcal{M}}$. We start by introducing indicator matrix \mathbf{X} and “goodness” function.

Let \mathbf{X} be a $k \times n$ binary match indicator matrix, where $\mathbf{X}(i, j) = 1$ means that the i^{th} query node in \mathcal{Q} matches the j^{th} node in \mathcal{G} ; and $\mathbf{X}(i, j) = 0$ otherwise. It can be seen that each row of \mathbf{X} has only one entry to be 1 and each column of \mathbf{X} has at most one entry to be 1. The match indicator matrix \mathbf{X} induces the matching subgraphs, i.e., the matching subgraph are the induced subgraph of \mathcal{G} whose corresponding columns in \mathbf{X} are not empty. The match indicator matrix \mathbf{X} can be found through the following “goodness” function ³.

$$g(X) = -\|\mathbf{X}\mathbf{A}\mathbf{X}' - \mathbf{A}_{\mathbf{q}}\|_F^2 + a \cdot \text{trace}(\mathbf{S}\mathbf{X}') - b \cdot \|\mathbf{X}\mathbf{X}' - \mathbf{I}\|_F^2 \quad (3.29)$$

where a and b are parameters that balance the weight of each term.

The idea behind the goodness function to calculate $X(k \times n)$ that best embodies both the rankings in the pair-wise similarity matrix (the second term) and the original connectivity consistency (the first term) of the network. In our developed method, we first drive the index matrix \mathbf{T} , which gives the node pairs that should be connected in the resulting matching subgraph; then convert all connected subgraphs to “super nodes” (i.e., connected node sets); and finally find the bridges between the corresponding super nodes. The developed procedure to find matching subgraph is summarized in Algorithm 3.10.

Since the counterpart of the query graph can be found in the indicator matrix \mathbf{X} , the connection among the counterparts should be decided. In line 1 and 2, the algorithm finds the indices of nodes that are directly connected and not connected in the original data

³In this work, we use a local heuristic to find \mathbf{X} from \mathbf{S} by searching the top- l entries in each row of \mathbf{S} , where l is a small number (e.g., $l = 3$).

Algorithm 3.10 SIM2SUB

Input: Indicator Matrix $X(k \times n)$, the data network \mathcal{G} , revised query graph $\tilde{\mathcal{Q}}$.

Output: The updated matching subgraph $\tilde{\mathcal{M}}$

- 1: $\mathbf{T} = \mathbf{X}'\mathbf{A}_q\mathbf{X} - \text{and}(\mathbf{A}, \mathbf{X}'\mathbf{A}_q\mathbf{X})$; //index matrix of nodes in subgraph that should be connected;
 - 2: Construct index I of connected nodes from $\text{and}(\mathbf{A}, \mathbf{X}'\mathbf{A}_q\mathbf{X})$;
 - 3: **for** each connected node set C in I **do**
 - 4: Construct "super node" C' ;
 - 5: **end for**
 - 6: Update the data network \mathcal{G} ;
 - 7: **for** each unconnected "super node" pair (C'_1, C'_2) from \mathbf{T} **do**
 - 8: Connect (C'_1, C'_2) by the shortest path;
 - 9: **end for**
 - 10: return $\tilde{\mathcal{M}}$;
-

network \mathcal{G} . From line 3 to line 10, the algorithm constructs super nodes and find shortest paths as bridges among counterparts that should be connected. The method generates one subgraph from one indicator matrix \mathbf{X} . If multiple results (e.g., t results) are required, then t indicator matrices which have top t largest "goodness" values will be constructed as described in Section 3.3.2.

Complexity Analysis. In this section we give the complexity analysis of the developed algorithms (i.e., FIRST-Q, FIRST-N and FIRST-E).

Lemma 3.6. Complexity of FIRST-Q & FIRST-N. The time complexity of Algorithm 3.7 and Algorithm 3.8 is $O(r^2t^2kn + rtkn + K^2kn)$, and its space complexity is $O(k^2rn + m_1)$. Here, n and k are the orders of the number of nodes of the input data network and the query graph, respectively; K denotes the number of unique node attributes, and r and t are the rank of eigendecomposition; m_1 is the number of edges in attributed network \mathcal{G} .

Proof. Firstly, since the diagonal degree matrix \mathbf{D} needs to be updated, from equation 3.21, \mathbf{D} is given as follows when edge attribute is missing:

$$\mathbf{D} = \text{diag}\left(\sum_{k,k'=1}^K \underbrace{(\mathbf{N}^k \mathbf{A} \mathbf{N}^{k'} \mathbf{1}) \otimes (\mathbf{N}_q^{k'} \mathbf{A}_q \mathbf{N}_q^k \mathbf{1})}_{O(k^2)+O(nk)=O(nk)}\right)^{O(k^2)} \quad (3.30)$$

This is because $k \ll n$ based on our assumption. Also note that during the updating, \mathbf{A} and \mathbf{N} of the network \mathcal{G} is unchanged. In all, the complexity of updating \mathbf{D} is $O(K^2kn)$. In the process of computing $\tilde{\mathbf{s}}$,

$$\tilde{\mathbf{s}} = \underbrace{(1 - \alpha)\mathbf{P}^{-1}[\mathbf{D}_1^{-1} + \alpha\mathbf{D}_1^{-1}\mathbf{L}]}_{O(r^2t^2kn)+O(rtn)} \underbrace{(\mathbf{\Lambda}^{-1} - \alpha\mathbf{R}\mathbf{D}_1^{-1}\mathbf{L})^{-1}}_{O(r^2t^2kn)+O(r^3)=O(r^2t^2kn)} \mathbf{R}\mathbf{D}_1^{-1}\mathbf{P}^{-1}\mathbf{h} \quad (3.31)$$

Secondly, when computing the above multiplication in backward way in order to make use of the linear complexity property of vector multiplication, the complexity can achieve $O(r^2t^2kn) + O(rtkn)$. The time for the rest of the computation in the algorithm is smaller, and thus can be ignored in the big-O notation. Overall, the time complexity of FIRST-Q is $O(r^2t^2kn + rtkn + K^2kn)$.

The proof of space complexity is omitted for space. QED.

Lemma 3.7. Complexity of FIRST-E. The time complexity of Algorithm 3.9 is $O(r^2t^2kn + Lrtkn + K^2Lkn)$, and its space complexity is $O(Lrtkn + m_1)$. Here, n and k are the orders of the number of nodes of the input network and query graph, respectively; K , L denotes the number of unique node and edge attributes, respectively and r , t are the rank of eigendecomposition; m_1 is the number of edges in attributed network \mathcal{G} .

Proof. When \mathbf{N} and \mathbf{E} are both available, in the process of updating diagonal degree matrix \mathbf{D} :

$$\begin{aligned} \mathbf{D} &= \text{diag}(\underbrace{\sum_{k,k'=1}^K \sum_{l=1}^L (\mathbf{N}^k (\mathbf{E}^l \odot \mathbf{A}) \mathbf{N}^{k'} \mathbf{1}) \otimes (\mathbf{N}_q^k (\mathbf{E}_q^l \odot \mathbf{A}_q) \mathbf{N}_q^{k'} \mathbf{1})}_{O(K^2Lkn)} \underbrace{\phantom{\sum_{k,k'=1}^K \sum_{l=1}^L (\mathbf{N}^k (\mathbf{E}^l \odot \mathbf{A}) \mathbf{N}^{k'} \mathbf{1}) \otimes (\mathbf{N}_q^k (\mathbf{E}_q^l \odot \mathbf{A}_q) \mathbf{N}_q^{k'} \mathbf{1})}}_{O(k^2)}) \quad (3.32) \\ \tilde{\mathbf{s}} &= \underbrace{(1 - \alpha)[\mathbf{I} + \alpha \mathbf{L} \quad \underbrace{(\mathbf{\Lambda}^{-1} - \alpha \mathbf{R} \mathbf{L})^{-1}}_{O(Lr^3t^3)} \quad \mathbf{R}]}_{O(r^2t^2kn + Lrtkn)} \mathbf{h} \end{aligned}$$

As we see from the above equations with the heaviest computation in the algorithm, the time complexity of FIRST-E is $O(r^2t^2kn + Lrtkn + K^2Lkn)$. The complexity of the rest of the computation in the algorithm can be reasonably ignored.

The proof of space complexity is omitted for space. QED.

3.3.3 Experimental Results

In this section, we present the experimental results and analysis of our developed algorithms. The experiments are designed to answer the following questions:

1. **Effectiveness.** How effective are our developed subgraph generating algorithms compared to other algorithms when different structures of queries are sent as inputs?
2. **Efficiency.** How fast are our developed algorithms compared to other techniques when they are applied on different size of real networks? How do our algorithms scale?

Experimental Setup. We first describe the experimental setup as follows.

Datasets. We use five different real-world datasets in our experiments, summarized in Table 3.5.

Table 3.5: Datasets Used in Evaluations

Name	# of Nodes	# of Edges	Node/Edge Attribute
<i>DBLP</i>	9,143	16,338	Node attribute only
<i>Flickr</i>	12,974	16,149	Node attribute only
<i>LastFm</i>	136,421	1,685,524	Node attribute only
<i>ArnetMiner</i>	1,274,360	4,756,194	Node & edge attribute
<i>LinkedIn</i>	6,726,290	19,360,690	Node attribute only

- *DBLP*: In the graph of *DBLP* each node represents an author who published paper in popular Data Mining and Database conferences and journals. Undirected edges represent co-authorship and each author has one attribute vector of the number of publications in 29 major conferences [128].
- *Flickr*: The graph of this dataset is the network of friends on *Flickr*. Node attribute vector is transformed from users’ profile information [129].
- *LastFm*: This dataset contains the following relationships of users on *LastFm* [129]. The node attribute vector is also transformed from users’ profile information, such as age, gender and location. It was collected in 2013.
- *ArnetMiner*: The graph in *ArnetMiner* dataset represents the academic social network. Undirected edges represent co-authorship and node attribute vector is extracted from number of published papers [129].
- *LinkedIn*: The graph of *LinkedIn* dataset is from users’ connection relationship in the social network in *LinkedIn*. The node attribute vector is transformed from users’ profile information such as age, gender and occupation, etc.

Comparison Methods. We compare our developed algorithms with *G-Ray* [105], *MAGE* [130], *FINAL* together with its variants (e.g., *FINAL-N*, *FINAL-N+*, *FINAL-NE*) [30]. To be specific, according to the two-phase nature of our method, we compare the similarity matrix calculated by our method and *FINAL* and the subgraph with *G-Ray* and *MAGE*. We also verify the efficiency of our developed method on five real-world datasets and test the scalability.

Machine. The following experiments are tested on 64-bit Windows Machine with 3.60 GHz CPU and 32.0 GB RAM. Programs are implemented in MATLAB with single thread.

Effectiveness.

Matching Graph Comparison. We evaluate the effectiveness of each of the two phases of the whole algorithm. First, we show how well our algorithms can perform on computing the cross-network node similarities. We define the distance of two similarity matrices computed by two different methods. The distance is calculated as the Frobenius norm of the difference between two similarity matrices: $distance = \| \mathbf{S} - \mathbf{S}' \|_F$, where \mathbf{S} is computed by *FINAL-N+* and \mathbf{S}' is computed by FIRST. The distance against the number of query nodes indicate the closeness between the similarity matrix returned from FIRST and *FINAL-N+*. As we observe from Figure 3.8, as the number of query nodes increase, the distance can be lower than 1.5×10^{-6} , which indicates that the similarity results computed by these two methods are quite close. Next we treat the similarity matrix returned from *FINAL-N+* as groundtruth and define the precision and recall to evaluate how well our method approximates *FINAL-N+*. First we sort both similarity matrix \mathbf{S} and \mathbf{S}' and truncate top k columns as retrieved results \mathbf{M} . The precision and recall are then calculated with regard to p ($p \leq k$) selected columns. For each row, if the entry in \mathbf{S}' of selected matrix is also in \mathbf{M} , then we consider it as relevant. From Figure 3.8, we can observe that the precision tends to be high when the recall is low. Also the overall tendency shows that a relatively small r leads to a high precision.

In the second phase, we test the effectiveness of our subgraph generating algorithm against *G-Ray* and *MAGE* respectively. We design five typical query patterns and load them into three algorithms. The performance of three algorithms are summarized in Table 3.7 and Table 3.8. We define the terminology in the table as follows.

According to the summary, the patterns returned by *G-Ray* deviate significantly from the query pattern. The returned subgraph is reasonable in several patterns such as E-star (83.3% exact matching nodes) and line (50% exact matching nodes). For other patterns like clique (FIRST 57.1% vs. *G-Ray* 25.0% exact matching nodes) and loop (FIRST 71.4% vs. *G-Ray* 27.3% exact matching nodes), our method performs better. Generally speaking, thanks to our formulation that considers both the topology and pairwise node similarity, when the inputs contain more diverse and complicate patterns, the results returned by our method tends to have a better balance on the subgraph structure and attribute matching. Overall, our method also outperforms *MAGE* when edge attribute is taken into consideration.

Table 3.6: Terminology Definition in Table 3.7 & 3.8

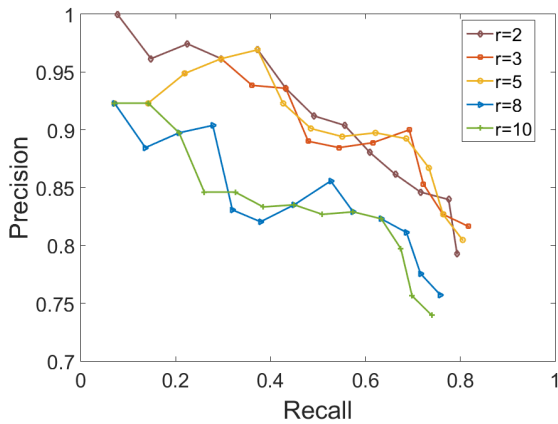
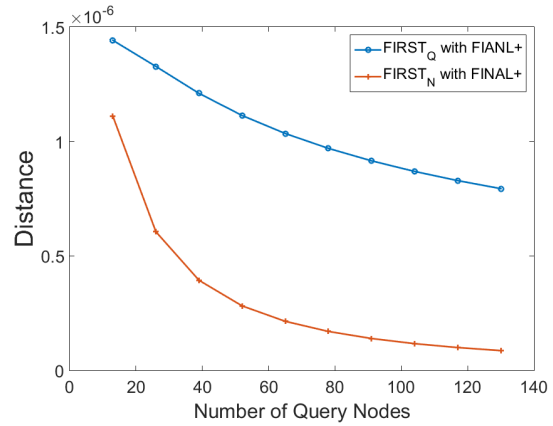
Name	Definition
<i>Extra Nodes</i>	Nodes in subgraph with incorrect node attribute or related position
<i>Exact Matching Nodes</i>	Nodes in subgraph with correct node attribute and related position
<i>Intermediate Nodes</i>	Nodes in the path between exact matching nodes
<i>Extra Edges</i>	Edges in subgraph with incorrect edge attribute or between extra nodes
<i>Exact Matching Edges</i>	Edges in subgraph with correct node attribute and between exact matching nodes
<i>Intermediate Edges</i>	Edges in the path between intermediate nodes

Table 3.7: Matching Comparison of 5 Patterns (Nodes)

Algorithm	% Extra Nodes			% Exact Matching Nodes			% Intermediate Nodes		
	G-Ray	MAGE	FIRST	G-Ray	MAGE	FIRST	G-Ray	MAGE	FIRST
Star(N)	62.5	*	0.0	37.5	*	75.0	0.0	*	25.0
E-Star(N)	0.0	*	0.0	83.3	*	71.4	16.7	*	28.6
Line(N)	50.0	*	0.0	50.0	*	83.3	0.0	*	16.7
Loop(N)	0.0	*	0.0	27.3	*	71.4	72.7	*	28.6
Clique(N)	25.0	*	0.0	25.0	*	57.1	50.0	*	42.9
Star(NE)	*	50.0	0.0	*	30.0	40.0	*	20.0	60.0
E-Star(NE)	*	0.0	0.0	*	33.3	41.7	*	66.7	58.3
Line(NE)	*	33.3	0.0	*	33.3	62.5	*	33.3	37.5
Loop(NE)	*	27.3	44.4	*	27.3	33.3	*	45.5	22.2
Clique(NE)	*	40.0	0.0	*	60.0	66.7	*	0.0	33.3

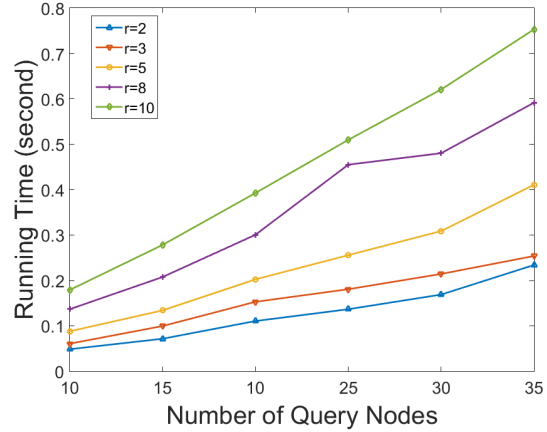
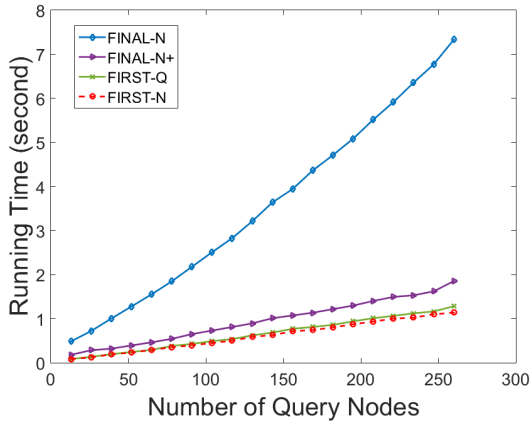
Table 3.8: Matching Comparison of 5 Patterns (Edges)

Algorithm	% Extra Edges			% Exact Matching Edges			% Intermediate Edges		
	G-Ray	MAGE	FIRST	G-Ray	MAGE	FIRST	G-Ray	MAGE	FIRST
Star(N)	66.7	*	0.0	33.3	*	57.1	0.0	*	42.9
E-Star(N)	0.0	*	0.0	60.0	*	50.0	40.0	*	50.0
Line(N)	60.0	*	0.0	40.0	*	60.0	0.0	*	40.0
Loop(N)	8.3	*	0.0	8.3	*	42.9	83.3	*	57.1
Clique(N)	35.7	*	0.0	7.1	*	12.5	64.3	*	87.5
Star(NE)	*	42.9	0.0	*	0.0	14.3	*	57.1	85.7
E-Star(NE)	*	0.0	0.0	*	7.1	9.0	*	92.9	91.0
Line(NE)	*	27.3	0.0	*	0.0	14.3	*	72.7	85.7
Loop(NE)	*	30.0	42.9	*	0.0	14.3	*	70.0	29.8
Clique(NE)	*	33.3	0.0	*	0.0	12.5	*	100.0	87.5

(a) Precision vs. Recall ($\alpha = 0.8$).

(b) Distance vs. Query Size.

Figure 3.8: Effectiveness Comparison (number of query nodes = 13, $r = 5$).



(a) Running time vs. Query Size comparison between *FINAL* and *FIRST*. ($\alpha = 0.8$).

(b) Running time vs. Query Size and Eigendecomposition Rank. ($\alpha = 0.8$).

Figure 3.9: Scalability Comparison and Analysis on Query Size and Eigendecomposition Rank.

Case Study. We also designed an interactive scenario which is shown in Figure 3.10. The

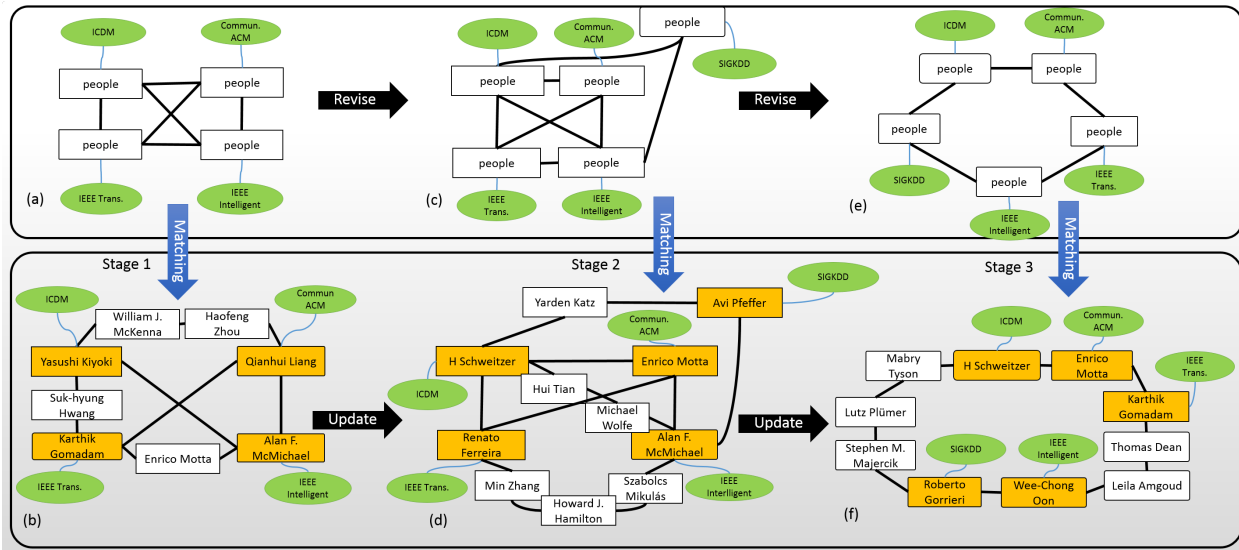


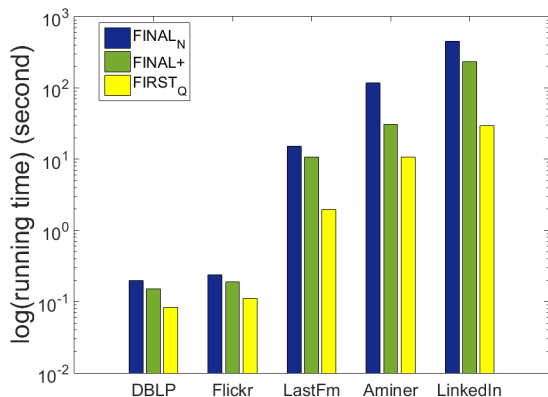
Figure 3.10: A case study of interactive attributed subgraph matching on *DBLP*. Best viewed in color. (a): query graphs, (b): matching subgraphs. Green ellipse: node attribute value, Yellow rectangle: matching node, White rectangle: extra/intermediate node.

experiment is conducted on *DBLP* dataset. The initial query is a 4-node clique which requests a group of co-authors from four different main conferences. The initial result gives an approximate clique with three intermediate authors. After that, the user adds one more author from SIGKDD to the group and the algorithm returns an approximate subgraph with six intermediate authors. Note that as the query is refined, the result is also incrementally

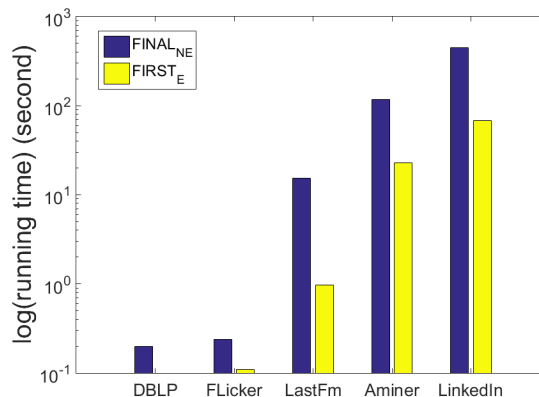
refined instead of a complete change, as Alan F. McMichael appears in both initial and updated result. This phenomena can be also observed from the following steps and it makes sense in the interactive procedure. As the user realizes that the structure is complex and incurs too many intermediate nodes, the query graph is revised to a loop fashion. The refined result shows H. Schweitzer and Enrico Motta in both results. Finally the user is shown a refined output with five intermediate nodes, and all five matching nodes are exact matching nodes from *DBLP*. This implies the cooperation pattern among the authors.

Efficiency. Lastly, we present the efficiency results for FIRST.

Speedup. We first evaluate the efficiency of our developed technique and there are two phases in this particular experiment. First, we only consider node attribute and perform FIRST against *FINAL-N* and *FINAL-N+* to compare the running time on five datasets. Then we add edge attribute and perform FIRST against *FINAL-NE*. In each test, the query graph is fixed and it is a relatively small graph with 13 nodes. The results are shown in Figure 3.11. From the result we can observe that our algorithm outperforms the other three algorithms with speedup from $2\times$ to $16\times$. Specifically, when the large graph has over 20M edges, The response time for exact network alignment method is too long to measure. But our method can incrementally update pair-wise similarity in about 20 seconds with a high accuracy (over 90%, see Section 3.3.3).



(a) Efficiency Comparison 1 ($\alpha = 0.8$).



(b) Efficiency Comparison 2 ($\alpha = 0.8$).

Figure 3.11: (Lower is better.) Log of Running time vs Datasets of different size. (number of query nodes = 13, $r = 5$).

Scalability. The scalability of FIRST is summarized in Figure 3.9 and they are tested on *DBLP*. We first measure the running time against the number of query nodes and also compare it with *FINAL-N* and *FINAL-N+*. We can see that the running time of FIRST grows linearly. Compared to *FINAL-N* and *FINAL-N+*, FIRST has better scalability as

the increase of query size. Next we measure the scalability with regard to the number of eigenvalues used in FIRST. We can see that for different r , the running time still grows linearly. At the point where there are 60 nodes and $r = 10$, the running time is still less than 1 second (0.8s). It shows that the quick response time of FIRST, which makes it suitable for interactive query feedback.

CHAPTER 4: PAIRWISE ASSOCIATION WITH NEURAL TECHNIQUES

In this chapter, we introduce the neural techniques which we have developed for the pairwise multi-network association problem. We first elaborate the Sylvester Multi-Graph Neural Network (SYMGNN framework), which is a neural generalization of the traditional Sylvester equation specifically for geometric matrix completion task. Second, we elucidate NEMOS model, a simplified multi-network GNN-based neural model for social recommendation, as an extension of the low-rank instantiation of SYMGNN framework.

4.1 SYLVESTER MULTI-GRAPH NEURAL NETWORK

As we have shown in Chapter 3, the Sylvester equation plays a central role for various applications in applied mathematics [136] [137], systems and control theory [138], machine learning [139] and graph mining [25]. Particularly in graph mining, the Sylvester equation has shown its applicability in various multi-network mining tasks, such as network alignment [30], subgraph matching [17], and social recommendation [15]. Despite its concise mathematical formulation and deep theoretical background on equation properties, there are several limitations when it is applied on multi-network mining. First, the real-world network data contains various heterogeneous features. However, the features of the networks can not directly be incorporated into the traditional Sylvester equation formulation. Second, in the task of multi-network association, the traditional Sylvester equation essentially calculates a linear transformation from the observed prior multi-network association matrix. However, the non-linear relation between the prior knowledge and the final solution can not be captured by the traditional Sylvester equation. Third, solving the Sylvester equation is decoupled from the downstream learning task, and it is not clear how to further adapt the solution of the Sylvester equation towards different multi-network mining tasks. For example, in network alignment task, the solution matrix of multi-network association is first calculated by the Sylvester equation. Then, the soft/hard alignment method is conducted on the solution, such as the greedy match. The Sylvester equation can not be trained or tuned in an end-to-end fashion as deep neural networks, and consequently the performance of the downstream tasks might be suboptimal. A natural question is: how can we obtain the best of both the traditional Sylvester equation and the neural networks?

In this chapter, we design a multi-graph neural network framework, SYMGNN in order to generalize the traditional linear Sylvester equation towards an end-to-end neural network model. Specifically, we focus on geometric matrix completion task, and elucidate two

instantiations for the SYMGNN framework. Our designed approach bears three distinctive advantages compared with both the Sylvester equation and the existing neural models targeted on geometric matrix completion. First, our framework is a general form, and it is flexible to be instantiated by different downstream tasks. Second, by leveraging the attention mechanism, our model incorporates the within network attention and cross-network attention, which strengthens the model expressiveness, and learns compatible node representations across different networks. Third, two instantiations are provided based on direct 2-dimensional multi-network association learning and low-dimensional representation learning for separate networks, respectively. The low-dimensional instantiation approach can potentially further reduce the model’s space complexity.

The notations used throughout the chapter are summarized in Table 4.1. Generally, we use bold uppercase letters to represent matrices, bold lowercase letters to represent vectors, lowercase or uppercase letters in regular font for scalars.

Table 4.1: Symbols and Definition

Symbols	Definition
$\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{F}_1\}$	a network with feature matrix
\mathbf{H}	prior knowledge matrix of cross-network associations
$\mathbf{D}_1, \mathbf{D}_2$	diagonal degree matrices
\mathbf{I}	an identity matrix
\mathbf{W}, Θ	learnable parameter matrices
α, β	the weighting parameter $0 < \alpha, \beta < 1$
d	the dimension of features
$\langle \mathbf{v}_i, \mathbf{v}_j \rangle$	the inner product of $\mathbf{v}_i, \mathbf{v}_j$
$\text{bmm}(\cdot)$	batch matrix multiplication
$\ \cdot\ _F$	Frobenius norm
$\text{diag}(\mathbf{v})$	construct a diagonal matrix by vector \mathbf{v}

Before giving the definition of GNN-based neural Sylvester equation, we first provide some preliminaries on the traditional Sylvester equation and the Graph Neural Networks, followed by a formal definition of the geometric matrix completion.

4.1.1 Preliminaries

Sylvester Equation for Multi-network Mining. Given two networks represented as $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{F}_1\}$, $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{F}_2\}$, and an anchor multi-network association matrix \mathbf{H} , which denotes the prior knowledge of the multi-network node associations. The Sylvester equation

for multi-network mining is defined as follows [25]:

$$\mathbf{X} = \alpha \tilde{\mathbf{A}}_2 \mathbf{X} \tilde{\mathbf{A}}_1^\top + (1 - \alpha) \mathbf{H} \quad (4.1)$$

where $\tilde{\mathbf{A}}_1$ and $\tilde{\mathbf{A}}_2$ are the symmetrically normalized adjacency matrices of the input networks. The \mathbf{X} represents the cross-network node association scores which the equation aims to calculate. The scalar $\alpha \in (0, 1)$ is aimed at weighting the multi-network association aggregation term (i.e. $\tilde{\mathbf{A}}_2 \mathbf{X} \tilde{\mathbf{A}}_1^\top$), and the prior knowledge term (\mathbf{H}). Due to the normalization of \mathbf{A}_1 and \mathbf{A}_2 , the corresponding linear system of Eq. (4.1) contains a positive semi-definite coefficient matrix, which guarantees the existence of unique solution for Eq. (4.1). Solving Eq. (4.1) is often time-consuming. A straightforward iterative method to solve Eq. (4.1) is the fixed point iteration. More efficient method is proposed in [25] with linear time and space complexity.

The formulation of this equation for multi-network mining (Eq. (4.1)) enjoys several distinctive advantages, which are summarized as follows. Firstly, theoretically the existence and uniqueness of the solution \mathbf{X} can be guaranteed. Furthermore, there exists various efficient and scalable solvers for the solution. Secondly, the solution \mathbf{X} can be seen as a fixed point of the equation and can be obtained by iteratively evaluating the Eq. (4.1). Compared to existing neural models, which might contain a number of hidden layers, there is no need to save the hidden states/representations. Thirdly, when reaching the fixed point, theoretically it is equivalent to proceed the recurrent process implied by Eq. (4.1) infinite times, so the formulation is able to leverage long-range dependency when solving \mathbf{X} .

However, despite the advantages and effectiveness in various tasks, generally there are also several limitations of this formulation which are summarized as follows. Firstly, the numerical features of the nodes can not be effectively utilized for calculating \mathbf{X} . Secondly, the \mathbf{X} can be seen as a linear transformation from the prior knowledge matrix \mathbf{H} . However, the potential non-linear relationship between them can not be captured by this formulation. Thirdly, since the equation is not learnable and not tunable, the solution \mathbf{X} should always be adapted to a target downstream task by another learning model, but not in an end-to-end fashion. This might result in suboptimal performance for the downstream task.

Graph Neural Networks. The Graph Neural Networks (GNN) are powerful deep learning models for network data. The basic idea of GNN model is to learn node representations via learnable aggregation, in which the node features are accumulated and transformed from the neighborhood features. Given a network $\mathcal{G} = (\mathbf{A}, \mathbf{F})$, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the adjacency matrix of \mathcal{G} , and $\mathbf{F} \in \mathbb{R}^{n \times d}$ is the feature matrix with d being the dimension of features,

representative GNN aggregation step at time step t can be written as follows.

$$\mathbf{X}^{(t+1)} = \phi(\tilde{\mathbf{A}}\mathbf{X}^{(t)}\mathbf{W} + \mathbf{\Omega}^{(t)}\mathbf{F}) \quad (4.2)$$

where $\tilde{\mathbf{A}}$ is the normalized adjacency matrix with added self-loops. \mathbf{W} is a learnable parameter matrix for the aggregated features. Different GNN models adopts different feature aggregation mechanisms. GCN model [140] inserts the adjacency matrix with self-links and applies the re-normalization. It also sets $\mathbf{\Omega} = \mathbf{0}$. GAT model [141] utilizes the self-attention mechanism in feature aggregation. GIN model [142] adopts an MLP layer after the aggregation of hidden representations of nodes for improving the discriminative ability of GNN model.

Convolutional Graph Embedding. Proposed in [143], the convolutional graph embedding (CGE) model is a GNN-based single network embedding model. Different from traditional GCN [140], which simply sums up the hidden representations of all neighbors, the aggregation weights for center nodes and neighbor nodes are differentiated and learnable with the model in CGE. Specifically, in the $(l + 1)$ -th layer, the output of a CGE layer can be represented as:

$$\mathbf{V}^{(l+1)} = \phi((\text{diag}(\boldsymbol{\sigma}) + (\mathbf{I} - \text{diag}(\boldsymbol{\sigma}))\tilde{\mathbf{A}})\mathbf{V}^{(l)}\mathbf{\Theta}^{(l)}) \quad (4.3)$$

where $\mathbf{V}^{(l+1)}$ and $\mathbf{V}^{(l)}$ are the node representation matrices of the $(l + 1)$ -th and l -th layer, respectively. $\boldsymbol{\sigma}$ is the learnable weight vector for the self-connections, and $\mathbf{\Theta}^{(l)}$ is the learnable weight matrix. $\phi()$ is an activation function. Using CGE adds more expressiveness to the model compared with GCN, and we will elaborate how to leverage it in Section 4.1.4.

4.1.2 Geometric Matrix Completion

Different from traditional matrix completion problem, the geometric matrix completion needs to handle two additional networks which reflect the topological relations between the nodes of two entities sets. Specifically, the problem is defined as follows.

Problem 4.1. GEOMETRIC MATRIX COMPLETION

Given: Two networks with node features $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{F}_1\}$, $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{F}_2\}$, and the partially observed multi-network association \mathbf{H} of the nodes in \mathcal{G}_1 and \mathcal{G}_2 ;

Output: The unobserved entries in \mathbf{H} .

In this section, we will elaborate our framework of implicit multi-graph neural networks. First, we will present a general framework of neural Sylvester equation, followed by two GNN-

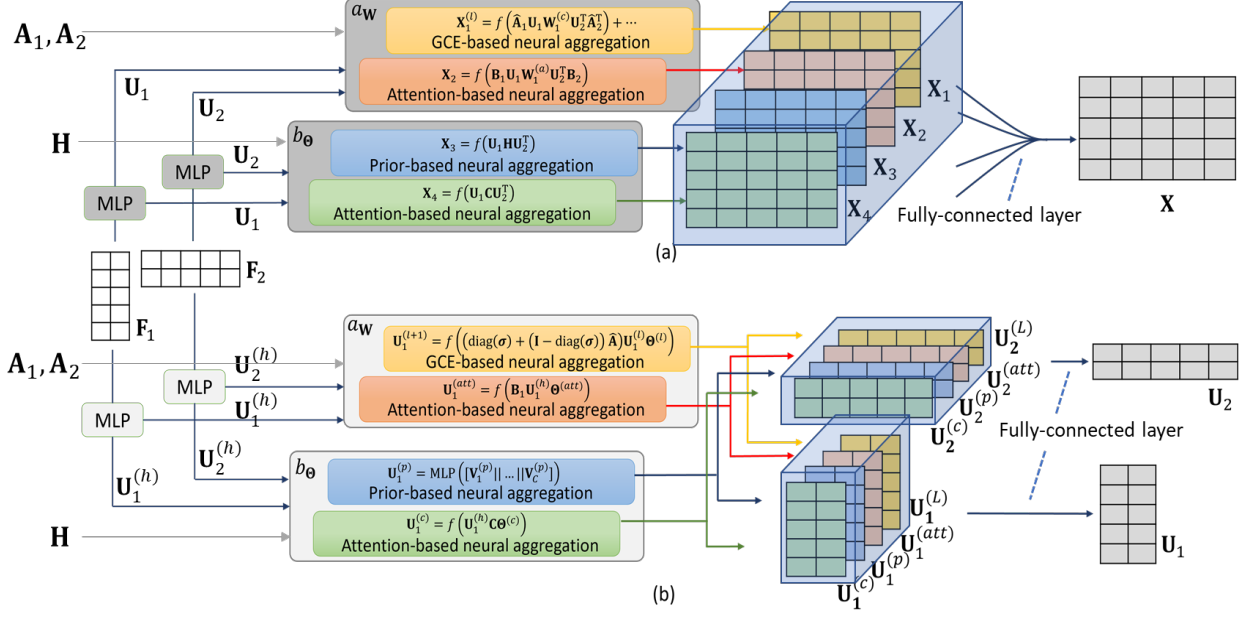


Figure 4.1: The overall illustration of two instantiations for SYMGNN. (a): the base model, and (b): the low-rank model.

based instantiations of the general framework targeted at the geometric matrix completion. Second, we will introduce the training method with details. Third, we will provide analysis of the two instantiations in terms of complexity.

4.1.3 Sylvester Multi-Graph Neural Network Framework

The goal of the proposed SYMGNN framework is to leverage the advantages of the traditional Sylvester equation, and in the meanwhile overcoming its limitations. First, if we observe the Sylvester equation in Eq. (4.1) from an iterative perspective, we can see that the first term on the right side aggregate the multi-network node association \mathbf{X} linearly for the updated \mathbf{X} . The second term incorporates the prior multi-network association message \mathbf{H} into the updated \mathbf{X} . Second, similar to the ideas of the traditional Sylvester equation, we identify the two key modules of the SYMGNN: (1) the multi-network aggregation learning module; and (2) the prior multi-network association incorporation learning module. The general framework can be represented in Eq. (4.4).

$$\mathbf{X} = \phi(\alpha \cdot a_{\mathbf{W}}(\mathbf{F}_1, \mathbf{F}_2, \tilde{\mathbf{A}}_1, \tilde{\mathbf{A}}_2) + (1 - \alpha) \cdot b_{\Theta}(\mathbf{F}_1, \mathbf{F}_2, \mathbf{H})) \quad (4.4)$$

where $a_{\mathbf{W}}()$ and $b_{\Theta}()$ are two neural modules with parameters \mathbf{W} and Θ , and weighting scalar $\alpha \in [0, 1]$. $\phi()$ is a non-linear activation function. \mathbf{X} is the multi-network association

output of the SYMGNN framework, and it can be further fed into a neural network for adapting towards a downstream task in an end-to-end fashion. As we can see, this framework is a neural generalization originated from the Sylvester equation in Eq. (4.1). When the neural modules $a_{\mathbf{W}}$ and b_{Θ} are linear aggregation functions, the Eq. (4.4) degenerates to the classic Sylvester equation Eq. (4.1). Therefore, we can say that the Sylvester Multi-Graph Neural Network framework is a non-linear and neural generalization of the traditional linear Sylvester equation. For Eq. (4.4), numerous instantiations exist for different downstream tasks. Next, we will discuss how to specifically instantiate this framework towards geometric matrix completion (Problem 4.1).

4.1.4 Base Model for Geometric Matrix Completion

Here, we present our base model for geometric matrix completion problem. In order to instantiate $a_{\mathbf{W}}(\mathbf{F}_1, \mathbf{F}_2, \mathbf{A}_1, \mathbf{A}_2)$, we design two parallel layers which adopt the CGE-based aggregation layer and the attention-based aggregation layer respectively. The motivation here is to learn the 2-d hidden representations for the multi-network association solution. In order to achieve this, we design two types of 2-d convolutional non-linear aggregation module, namely the adjacency matrix-based GCE neural aggregation, and the attention-based neural aggregation. To be specific, given $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{F}_1\}$, $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{F}_2\}$ with n_1, n_2 nodes respectively, we first apply the learnable parameters of self-connections from Eq. (4.3) on \mathbf{A}_1 and \mathbf{A}_2 to obtain the updated adjacency matrices. The goal is to adopt the learnable weight of the self-connections from CGE for improving the expressiveness of the model:

$$\hat{\mathbf{A}}_1 = \text{diag}(\boldsymbol{\sigma}_1) + (\mathbf{I} - \text{diag}(\boldsymbol{\sigma}_1))\tilde{\mathbf{A}}_1 \quad (4.5a)$$

$$\hat{\mathbf{A}}_2 = \text{diag}(\boldsymbol{\sigma}_2) + (\mathbf{I} - \text{diag}(\boldsymbol{\sigma}_2))\tilde{\mathbf{A}}_2 \quad (4.5b)$$

where $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ are learnable weights for self-connections of the first network and the second network respectively. Before applying the multi-network aggregation layers, the node features of \mathcal{G}_1 and \mathcal{G}_2 are fed into an MLP layer for obtaining the hidden features $\mathbf{U}_1 = \text{MLP}_1(\mathbf{F}_1)$ and $\mathbf{U}_2 = \text{MLP}_2(\mathbf{F}_2)$. In the CGE-based multi-network aggregation, the output of the l -th level aggregation can be represented as:

$$\mathbf{X}_1^{(l)} = \sum_{i=1}^l \phi(\hat{\mathbf{A}}_1^i \mathbf{U}_1 \mathbf{W}_i^{(c)} \mathbf{U}_2^{\top} (\hat{\mathbf{A}}_2^i)^{\top}) \quad (4.6)$$

where $\mathbf{W}_1^{(c)}, \dots, \mathbf{W}_l^{(c)}$ are parameter matrices and $\phi(\cdot)$ is an activation function. In practice, $\mathbf{W}_i^{(c)}, i = 1, 2, \dots, l$ is implemented as $\mathbf{W}_i^{(c)} = \mathbf{W}'_i(\mathbf{W}'_i)^\top$, as a metric learning approach for the generalization of Mahalanobis distance [144], in order to capture the feature correlation between nodes from two different networks.

In the attention-based multi-network aggregation, the output can be represented as:

$$\mathbf{X}_2 = \phi(\mathbf{B}_1 \mathbf{U}_1 \mathbf{W}^{(a)} \mathbf{U}_2^\top \mathbf{B}_2^\top) \quad (4.7)$$

where $\mathbf{W}^{(a)}$ is the parameter matrix, \mathbf{B}_1 and \mathbf{B}_2 are attention score matrices for \mathcal{G}_1 and \mathcal{G}_2 respectively. For instance, the attention score of node $(i, j) \in \mathcal{G}_1$ is calculated as:

$$\mathbf{B}_1(i, j) = \frac{\exp(\langle \mathbf{u}_i, \mathbf{u}_j \rangle)}{\sum_{k=1}^{n_1} \exp(\langle \mathbf{u}_i, \mathbf{u}_k \rangle)} \quad (4.8)$$

In order to instantiate $b_{\Theta}(\mathbf{F}_1, \mathbf{F}_2, \mathbf{H})$, similar to the first term $a_{\mathbf{W}}(\mathbf{F}_1, \mathbf{F}_2, \tilde{\mathbf{A}}_1, \tilde{\mathbf{A}}_2)$, we can also adopt two types of parallel aggregation layers. The first one is the direct neural aggregation from prior multi-network association, and the second one is via attention schema. However, since the entries of the prior multi-network association matrix \mathbf{H} is often real values or multi-class categorical rates, it is unreasonable to directly use \mathbf{H} for cross-network feature aggregation. Thus the prior multi-network association-based multi-network aggregation is only adopted when \mathbf{H} contains binary associations. The two types of multi-network aggregation modules are shown as follows.

$$\mathbf{X}_3 = \phi(\mathbf{U}_1 \mathbf{H} \mathbf{U}_2^\top) \quad (4.9a)$$

$$\mathbf{X}_4 = \phi(\mathbf{U}_1 \mathbf{C} \mathbf{U}_2^\top) \quad (4.9b)$$

where the cross-network attention score matrix \mathbf{C} is calculated as $\mathbf{C}(i, j) = \frac{\exp(\langle \mathbf{u}_i, \mathbf{u}_j \rangle)}{\sum_{k=1}^{n_2} \exp(\langle \mathbf{u}_i, \mathbf{u}_k \rangle)}$ for $i \in \mathcal{G}_1, j \in \mathcal{G}_2$.

Putting everything together, as shown in Figure 4.1, the intermediate multi-network association matrices $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4$ consist of the hidden representation tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times 4}$ for the multi-network association solution. We apply a fully connected layer on \mathcal{X} for obtaining the final multi-network association $\mathbf{X} = \text{bmm}(\mathcal{X}, \mathcal{W})$ where $\mathcal{W} \in \mathbb{R}^{n_1 \times 4 \times 1}$ is the parameter tensor.

4.1.5 Low-rank Model for Geometric Matrix Completion

Instead of conducting bi-linear neural aggregation for the multi-network association directly, we can generate the embeddings for nodes of \mathcal{G}_1 and \mathcal{G}_2 respectively. Similar to

the base model, we consider both the direct neural aggregation from the original network topology, and the neural aggregation from the within network attentions. First, given two networks $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{F}_1\}, \mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{F}_2\}$ with n_1, n_2 nodes respectively, the node features are fed into an MLP layer for obtaining the hidden features $\mathbf{U}_1^{(h)} = \text{MLP}_1^{(l)}(\mathbf{F}_1)$ and $\mathbf{U}_2^{(h)} = \text{MLP}_2^{(l)}(\mathbf{F}_2)$. Similar to the motivation of the base model, $\mathbf{U}_1^{(h)}, \mathbf{U}_2^{(h)}$ are then fed into two parallel CGE-based and attention-based neural modules for generating the hidden representations of the node features for two networks separately. We take $\mathbf{U}_1^{(h)}$ as an example, and the process for $\mathbf{U}_2^{(h)}$ is similar. The updated node hidden representations after an l -layer CGE module is:

$$\mathbf{U}_1^{(l+1)} = \phi((\text{diag}(\boldsymbol{\sigma}) + (\mathbf{I} - \text{diag}(\boldsymbol{\sigma}))\tilde{\mathbf{A}})\mathbf{U}_1^{(l)}\mathbf{W}^{(l)}) \quad (4.10)$$

where $\mathbf{U}_1^{(0)} = \mathbf{U}_1^{(h)}$, $\boldsymbol{\Theta}^{(l)}$ is the parameters for the l -th layer, and $\phi(\cdot)$ is an activation function. After L layers, we obtain $\mathbf{U}_1^{(L)}$. The updated node hidden representations after the attention-based neural aggregation module is:

$$\mathbf{U}_1^{(att)} = \phi(\mathbf{B}_1\mathbf{U}_1^{(h)}\mathbf{W}^{(att)}) \quad (4.11)$$

where the attention score matrix \mathbf{B}_1 can be calculated via Eq. (4.8). $b_{\boldsymbol{\Theta}}(\mathbf{F}_1, \mathbf{F}_2, \mathbf{H})$ is also instantiated for \mathcal{G}_1 and \mathcal{G}_2 separately. Here, we can adopt a similar prior multi-network association-based neural aggregation when the prior \mathbf{H} denotes binary or multi-class relations. For \mathbf{H} with entries of K classes, we apply K neural networks, in which each neural network aggregates one class of nodes.

$$\mathbf{V}_i^{(p)} = \phi(\mathbf{H}_i\mathbf{U}_1^{(h)}\boldsymbol{\Theta}_i^{(p)}) \quad (4.12a)$$

$$\mathbf{U}_1^{(c)} = \phi(\mathbf{C}\mathbf{U}_1^{(h)}\boldsymbol{\Theta}^{(c)}) \quad (4.12b)$$

where \mathbf{H}_i is the prior multi-network association which only contains the entries of the i -th class. $\boldsymbol{\Theta}_i^{(p)}$ and $\boldsymbol{\Theta}^{(c)}$ are learnable parameters. The $\mathbf{V}_i^{(p)}$ for all classes are then concatenated and fed into an MLP for the node representation $\mathbf{U}_1^{(p)} = \text{MLP}([\mathbf{V}_1^{(p)} || \dots || \mathbf{V}_K^{(p)}])$. The cross-network attention matrix \mathbf{C} is calculated by the same method as in Eq. (4.9b).

Putting everything together, we now have four representation matrices for each network: $\mathbf{U}_1^{(L)}, \mathbf{U}_1^{(att)}, \mathbf{U}_1^{(p)}, \mathbf{U}_1^{(c)}$. We can adopt another fully connected layer to obtain a final representation \mathbf{U}_1^1 . The predicted multi-network association between two nodes is calculated by

¹We find that by simply adding them with the original node hidden representations, we can already achieve superior performance.

the dot product of the row vectors of the resulting representation matrices \mathbf{U}_1 and \mathbf{U}_2 .

4.1.6 Training

For matrix completion, we adopt the Mean Squared Error (MSE) loss for both instantiations:

$$\mathcal{L}_1 = \|\mathbf{H} - \mathbf{M} \odot \mathbf{X}\|_F^2 \quad (4.13a)$$

$$\mathcal{L}_2 = \|\mathbf{H} - \mathbf{M} \odot (\mathbf{U}_1 \mathbf{U}_2^T)\|_F^2 \quad (4.13b)$$

where the \mathbf{M} matrix is a mask of 0, 1, with 1 indicating the position of the observed prior multi-network associations. For the low-rank instantiation, the dot product of the node representations are used as the final solutions. For the regularization of the model parameters, we adopt the weight decay method with 0.01 as decay factor as we find that it shows slightly better performance over L_2 regularization. We use the Adam optimizer as it overall shows the stablest training.

4.1.7 Complexity Analysis

For notation simplicity, assume that the two input networks contain n nodes and m edges respectively. Suppose the feature dimension is d , the number of observed rating is m' and the dimension of node representations is $r < d$. For the base model, the major computation lies in the within-network and cross-network attention calculation as well as the aggregation. From Eq. (4.8), the within-network attention aggregation costs $O(n^2d)$. From Eq. (4.9b), the cross-network attention aggregation costs $O(n^2d)$. Eq. (4.3) costs $O(Lmd + d^2n)$. Since usually $r, d \ll m, n$, so its computation is not comparable with the attention-based neural aggregation. The overall time complexity for the base model is $O(\#iter \cdot (n^2d))$, where $\#iter$ is the total number of iterations. The space complexity is $O(n^2)$ because of the main storage of attention score matrices. Similarly, for the low-rank model, the overall time and space complexity are also $O(\#iter \cdot (n^2d))$ and $O(n^2)$. However, for the low-rank model, if we do not apply the within-network and cross-network attention-based neural aggregation, the time and space complexity would be reduced to $O(L(md + nd^2) + m'd)$, and $O(m + n(d^2 + r^2))$ respectively. Since the base model needs to store the intermediate multi-network association matrix, the space complexity can not be further reduced even if the attention-based neural aggregation is dropped.

In the next section, we present the experimental results on real-world benchmark datasets to show the effectiveness of the instantiations of the SYMGNN models.

4.1.8 Experimental Setting

Dataset and Pre-processing. The benchmark datasets used in the experiments are summarized in Table 4.2. For the benchmark datasets, ML-3K and Flixster have both user-user

Table 4.2: The statistics of the benchmark datasets.

Dataset	# of Users	# of Items	# of Ratings	Density
Douban	3,000	3,000	136,891	0.0152
Flixster	3,000	3,000	26,173	0.0029
YahooMusic	3,000	3,000	5,335	0.0006
ML-100K	943	1,682	100,000	0.0630
ML-1M	69,878	10,677	1,000,209	0.0447

and item-item interaction network. Douban only contains a user-user interaction network and YahooMusic only contains a item-item interaction network. For these two datasets, we use the identity matrix as the adjacency matrix for the missing networks. For ML-100K, ML-1M, we construct their user-user and item-item interaction network by adopting a k-nearest neighbors search via their features, and the k is treated as a hyperparameter in our model. All the datasets include multi-class categorical ratings. For the training/testing split, we use the same partition which is also adopted by existing methods, such as [143] [34], etc.

Baseline Methods. We use five baselines in our comparison, including the traditional Sylvester equation *Sylv.* [137], and recent neural network-based and GNN-based methods: *IGMC* [41], *GC-MC* [42], *PinSage* [92], and *sRGCNN* [34].

Experimental and Hyperparameter Settings. For the effectiveness comparison, we tune the hyperparameters of the model based on the best performance on the validation set. We use 2-layer GCE and attention aggregation in both instantiations on all datasets except for ML-100K and ML-1M. On these two datasets, the base model uses 3-layer GCE and attention aggregation. For the k-NN method used for generating social networks and item-item interaction networks on ML-100K and ML-1M dataset, we use $k = 10$ for the low-rank model and $k = 12$ for the base model. Further studies of the sensitivity of k will be discussed in the ablation study. The metric for comparison is the widely adopted rooted mean squared error (RMSE).

4.1.9 Effectiveness Results

The first comparison results are shown in Table 4.3, as these datasets are the most common datasets among all existing methods. The results are reported based on the average of five runs. The best performances are shown in bold fonts and the second best performances

are shown with underlines. As we can observe from the table, the traditional Sylvester equation can not achieve competitive results compared to other neural network/GNN-based baseline methods, which is consistent with our discussion on the limitations of the Sylvester equation. The Sylvester equation can not effectively incorporate node features, and also can not capture non-linear relations between the observed multi-network association and the solution. Among all the neural network-based methods, the proposed framework with low-rank instantiation outperforms the rest of the baselines on Douban, Flixster and YahooMusic datasets. Flixster (U) represents the dataset with only the usage of user-user interaction network. The performance of the proposed method slightly drops, and it shows the importance of both interaction networks of users and items in our model. Particularly, on YahooMusic dataset, the proposed method achieves 7.65% improvement over the best baseline. The average improvement over all datasets on Douban, Flixster and YahooMusic is 2.58%, which shows the effectiveness of the proposed models.

Table 4.3: RMSE comparison for geometric matrix completion.

Method	Douban	Flixster	Flixster (U)	Yahoo
Sylv.	1.220	1.244	1.276	29.403
IGMC	<u>0.729</u>	<u>0.895</u>	0.895	19.292
GC-MC	0.734	0.917	0.941	20.501
PinSage	0.739	0.954	0.951	22.954
sRGCNN	0.801	0.926	1.179	22.415
Ours (base)	0.762	0.911	0.934	<u>19.277</u>
Ours (low-rank)	0.725	0.891	<u>0.916</u>	17.815

The comparison results on ML-100K and ML-1M datasets are shown in Table 4.4. As we can see, the Sylvester equation is still not competitive with the rest of the methods. Our proposed low-rank instantiation consistently performs the best over all baselines. Among all baselines, *GC-MC* has close performance compared with our methods. *GC-MC* contains the graph encoder and the bi-linear decoder architecture which has similar effects to our proposed GNN-based neural aggregation model. This is consistent with our intuition of the effectiveness of the cross-network feature aggregation.

4.1.10 Ablation Study

The ablation study results are shown in Table 4.5. The 'Base model (G)' and 'Base model (A)' represent the model with only GCE-based neural aggregation and the model with only attention-based neural aggregation. The low-rank model uses the same abbreviation. The

Table 4.4: RMSE comparison on ML-100K and ML-1M dataset.

Method	ML-100K	ML-1M
Sylv.	1.403	1.323
IGMC	0.922	0.857
GC-MC	<u>0.905</u>	0.854
PinSage	0.942	0.906
sRGCNN	0.931	0.865
Ours (base)	0.915	<u>0.851</u>
Ours (low-rank)	0.899	0.843

values inside the parentheses denote the maximum allocated GPU memory in one epoch, in which we use the same batch size (i.e. 50) for comparison. As we can see, firstly the original model performs the best over all variants in terms of RMSE for both base and low-rank instantiations. Secondly, the model without the attention neural aggregation overall consumes the least GPU memory during training. On average, with only 1.24% performance drop, the models without attention neural aggregation show 16.98% less memory consumption. Furthermore, comparing with other baselines’ performance in Table 4.4, the variant low-rank model in Table 4.5 still outperforms all baseline methods.

Table 4.5: Ablation study on ML-100K and ML-1M dataset.

Method	ML-100K	ML-1M
Base model	0.915 (160Mb)	0.851 (2,371Mb)
Base model (G)	0.932 (141Mb)	0.862 (2,099Mb)
Base model (A)	0.924 (148Mb)	0.861 (2,209Mb)
Low-rank	0.899 (136Mb)	0.843 (1,983Mb)
Low-rank (G)	0.902 (110Mb)	0.857 (1,476Mb)
Low-rank (A)	0.920 (116Mb)	0.853 (1,641Mb)

4.1.11 Parameter Sensitivity

We mainly study the impact of the number of GCE-based neural aggregation layers and the k value for k-NN method in graph construction in ML-100K datasets. The results are shown in Figure 4.2 and Figure 4.3. From Figure 4.2, we can see that the performance is relatively stable for both models in terms of different number of layers. From Figure 4.3, the original models exhibit stabler performance over the models without the attention-based neural aggregation (the dashed lines) w.r.t. the k value. This can also indicate that the attention-based neural aggregation can make the model less sensitive to the hyperparameter

for constructing graphs when the user-user/item-item interactions are not directly available.

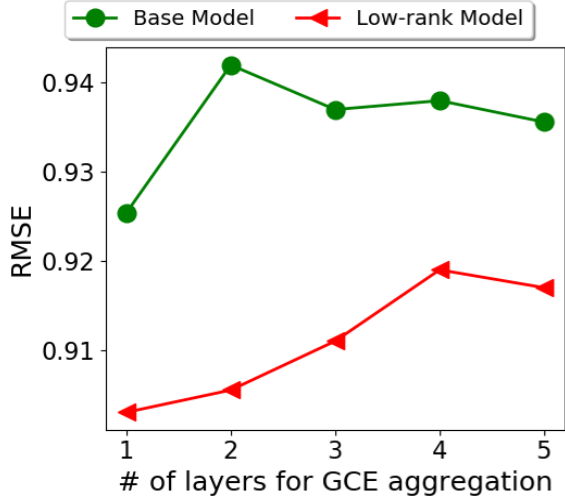


Figure 4.2: RMSE vs. number of layers for GCE aggregation.

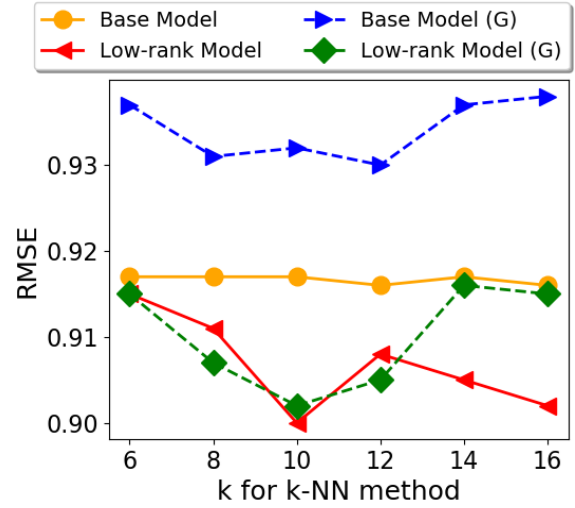


Figure 4.3: RMSE vs. k for k-NN method in graph construction.

4.2 A UNIFIED VIEW OF (NEURAL) SYLVESTER EQUATION MODEL

In this section, we provide a unified view of the traditional Sylvester equation introduced in Chapter 3 and the neural Sylvester equation model introduced in Chapter 4 for pairwise multi-network association in a framework of constrained optimization formulation. Before diving into the unified view, we first introduce a special type of non-linear equation, which is the preliminaries of the unified framework, and is closely related to the SYMGNN model.

4.2.1 Discussion on the Multi-network Equilibrium Equation

In this subsection, we elaborate the relationship between the developed SYMGNN model with a non-linear equation named equilibrium equation in multi-network scenario [145], and discuss a potential future direction. From Eq. (4.4), if the right side of the equation also takes the solution \mathbf{X} as an input, the resulting equation can be seen as a multi-network implicit neural model, in which the solution can be seen as the output of a neural model without the storage of hidden states. The equation is also referred to as the equilibrium equation, with the following formation:

$$\mathbf{X} = \phi(\alpha * a_{\mathbf{W}}(\mathbf{X}, \mathbf{A}_1, \mathbf{A}_2) + (1 - \alpha) * b_{\Omega}(\mathbf{F}_1, \mathbf{F}_2)) \quad (4.14)$$

where $\phi(\cdot)$ is a non-linear activation function. $a_{\mathbf{W}}(\mathbf{X}, \mathbf{A}_1, \mathbf{A}_2)$ is the generalized and learnable aggregation function for \mathbf{X} , and $b_{\Omega}(\mathbf{F}_1, \mathbf{F}_2)$ is the learnable function for incorporating numerical node features in updating \mathbf{X} . $\mathbf{F}_1, \mathbf{F}_2$ are node feature matrices for the input networks. α is a weighting scalar for balancing the weight between $a_{\mathbf{W}}(\cdot)$ and $b_{\Omega}(\cdot)$. The intuition behind Eq. (4.14) follows the idea of the Sylvester equation in Eq. (3.2). The first term $a_{\mathbf{W}}(\cdot)$ serves as the *propagation* term which propagates the current cross-network association to the neighborhoods. The second term $b_{\mathbf{W}}(\cdot)$ serves as the *linear transformation* term which directly transform the representation of one network to the other. Compared with Eq. (3.2), Eq. (4.14) can be seen as a generalized and learnable neural equation, with various potential model instantiations catering to different downstream tasks.

Eq. (4.14) implicitly defines the hidden state of \mathbf{X} so that \mathbf{X} does not need to be explicitly calculated at each layer of a recurrent process. As we will see shortly, under certain conditions (i.e. the well-posedness condition), the equation will be guaranteed to have a unique solution by iterating until reaching a fixed point. The conditions are discussed next.

4.2.2 The First Instantiation for Dense Solution

Here we propose one instantiation of the general framework of Eq. (4.14) for the dense solution matrix \mathbf{X} . The key ideas which tackle the limitations of the traditional Sylvester equation are two-fold. First, we observe that the aggregation term in Eq. (3.2) (i.e. $\mathbf{A}_2 \mathbf{X} \mathbf{A}_1^{\top}$) assigns equal weights for cross-network node pairs during aggregation, thus it could not differentiate between important and less important cross-network node pairs for the learning task. Our idea is to re-weight such node pairs in $a_{\mathbf{W}}(\cdot)$ in order to be more adaptive towards the downstream task. Second, we propose a cross-network distance metric learning function for $b_{\Omega}(\cdot)$, so that $b_{\Omega}(\cdot)$ provides an update to solution matrix by the feature space, which can not be obtained by cross-network association aggregation alone. Our proposed instantiation is as follows.

$$a_{\mathbf{W}}(\mathbf{X}, \mathbf{A}_1, \mathbf{A}_2) = (\mathbf{W}_2 \odot \mathbf{A}_2) \mathbf{X} (\mathbf{W}_1 \odot \mathbf{A}_1)^{\top} \quad (4.15a)$$

$$b_{\Omega}(\mathbf{U}_1, \mathbf{U}_2) = \sqrt{\mathbf{U}_2 \Omega \mathbf{U}_1^{\top}} \quad (4.15b)$$

where $\mathbf{W}_1, \mathbf{W}_2$ are parameter matrices which adjust the weights of neighbors when aggregating cross-network similarities. Here since \mathbf{A}_1 and \mathbf{A}_2 are typically sparse matrices, $\mathbf{W}_1, \mathbf{W}_2$ could have the same sparsity as \mathbf{A}_1 and \mathbf{A}_2 . As we can see the aggregation term in Eq. (3.2) becomes a special case where $\mathbf{W}_1 = \mathbf{W}_2 = \mathbf{I}$. Ω is a bi-linear parameter matrix. When Ω is a positive semi-definite matrix (i.e. $\Omega = \mathbf{W}^{(u)} (\mathbf{W}^{(u)})^{\top}$), $b_{\Omega}(\cdot)$ can be seen as a distance metric which adopts a generalized Mahalanobis distance [146].

Next, we will discuss the well-posedness conditions by which the instantiated equation (Eq. (4.14) by plugging in Eq. (4.15a) and Eq. (4.15b)) exists a unique solution.

Lemma 4.1. $\|\mathbf{W}_1\|_\infty < \lambda_{pf}(\mathbf{A}_1)^{-1}$ and $\|\mathbf{W}_2\|_\infty < \lambda_{pf}(\mathbf{A}_2)^{-1}$ are the sufficient conditions for the well-posedness of Eq. (4.14) when instantiated by Eq. (4.15a) and Eq. (4.15b).

Proof. Eq. (4.14) can be written as the following equivalent vectorized equation (weighting scalar α is absorbed in the matrix for notation brevity):

$$vec(\mathbf{X}) = \phi((\mathbf{W}_1 \odot \mathbf{A}_1) \otimes (\mathbf{W}_2 \odot \mathbf{A}_2))vec(\mathbf{X}) + vec(\mathbf{B}) \quad (4.16)$$

where $vec(\mathbf{X})$ is the vectorized matrix \mathbf{X} , and $\mathbf{B} = b_{\Omega(\mathbf{U}_1, \mathbf{U}_2)}$. By Lemma B.1 in [145], a sufficient condition for well-posedness is $\lambda_{pf}(|(\mathbf{W}_1 \odot \mathbf{A}_1) \otimes (\mathbf{W}_2 \odot \mathbf{A}_2)|) < 1$, where λ_{pf} is the Perron–Frobenius eigenvalue. Practically, we would need to resort to more tractable conditions during training. Specifically, $\lambda_{pf}(|(\mathbf{W}_1 \odot \mathbf{A}_1) \otimes (\mathbf{W}_2 \odot \mathbf{A}_2)|) = \lambda_{pf}(|\mathbf{W}_1 \otimes \mathbf{W}_2| \odot (\mathbf{A}_1 \otimes \mathbf{A}_2|))$, and we have the following strict condition, i.e. $\lambda_{pf}(|(\mathbf{W}_1 \otimes \mathbf{W}_2) \odot (\mathbf{A}_1 \otimes \mathbf{A}_2)|) \leq \lambda_{pf}(|\mathbf{W}_1 \otimes \mathbf{W}_2|)\lambda_{pf}(\mathbf{A}_1 \otimes \mathbf{A}_2) < 1$. So that the condition becomes:

$$\lambda_{pf}(|\mathbf{W}_1 \otimes \mathbf{W}_2|) < \lambda_{pf}(\mathbf{A}_1 \otimes \mathbf{A}_2)^{-1} \quad (4.17)$$

In practice, the matrix $\mathbf{W}_1 \otimes \mathbf{W}_2$ will need to be calculated as well as its Perron–Frobenius eigenvalue. However, this condition is not tractable. We use a more strict condition by observing that $\lambda_{pf}(|\mathbf{W}_1 \otimes \mathbf{W}_2|) \leq \|\mathbf{W}_1 \otimes \mathbf{W}_2\|_\infty = \|\mathbf{W}_1\|_\infty \|\mathbf{W}_2\|_\infty$, and let

$$\|\mathbf{W}_1\|_\infty \|\mathbf{W}_2\|_\infty < \lambda_{pf}(\mathbf{A}_1 \otimes \mathbf{A}_2)^{-1} \quad (4.18)$$

as the updated sufficient condition. Because $\lambda_{pf}(\mathbf{A}_1 \otimes \mathbf{A}_2) = \lambda_{pf}(\mathbf{A}_1)\lambda_{pf}(\mathbf{A}_2)$, and also $\|\mathbf{W}_1\|_\infty, \|\mathbf{W}_2\|_\infty \geq 0$, $\lambda_{pf}(\mathbf{A}_1)^{-1}, \lambda_{pf}(\mathbf{A}_2)^{-1} > 0$, when we have $\|\mathbf{W}_1\|_\infty < \lambda_{pf}(\mathbf{A}_1)^{-1}$ and $\|\mathbf{W}_2\|_\infty < \lambda_{pf}(\mathbf{A}_2)^{-1}$, the sufficient condition in Eq. (4.18) will be satisfied. QED.

This sufficient condition indicates that the \mathbf{W}_1 and \mathbf{W}_2 should lie in the hyper-ball of two different sizes. By the Theorem 2.2 in [147], if condition in Eq. (4.17) is satisfied, there always exists a re-scaled model that have the exact same output for the same input.

4.2.3 The Second Instantiation for Low-rank Solution

The idea of a low-rank representation model is to replace the $\mathbf{X} \leftarrow \mathbf{Y}\mathbf{Z}$, where \mathbf{Y} and \mathbf{Z} are low-rank representations of the node in \mathcal{G}_1 and \mathcal{G}_2 , respectively.

$$\mathbf{Y} = \phi(\alpha \mathbf{W}_a \mathbf{Y} (\mathbf{W}_2 \odot \mathbf{A}_2) + (1 - \alpha) \mathbf{W}_b \mathbf{U}_2) \quad (4.19a)$$

$$\mathbf{Z} = \phi(\beta \mathbf{W}_a \mathbf{Z}(\mathbf{W}_1 \odot \mathbf{A}_1) + (1 - \beta) \mathbf{W}_b \mathbf{U}_1) \quad (4.19b)$$

The well-posedness condition of the equilibrium equations Eq. (4.19a) and Eq. (4.28) is given in the following lemma.

Lemma 4.2. $\|\mathbf{W}_2\|_\infty \leq 1$ and $\|\mathbf{W}_a\|_\infty \leq \lambda_{pf}(\mathbf{A}_2)^{-1}$ are the sufficient conditions for the well-posedness of Eq. (4.19a). $\|\mathbf{W}_1\|_\infty \leq 1$ and $\|\mathbf{W}_b\|_\infty \leq \lambda_{pf}(\mathbf{A}_1)^{-1}$ are the sufficient conditions for the well-posedness of Eq. (4.28).

Proof. For Eq. (4.19a), similar to Lemma 4.1, a sufficient condition for well-posedness is $\lambda_{pf}(|(\mathbf{W}_2 \odot \mathbf{A}_2) \otimes \mathbf{W}_a|) < 1$. In order to have a more tractable condition, we have $\lambda_{pf}(|(\mathbf{W}_2 \odot \mathbf{A}_2) \otimes \mathbf{W}_a|) = \lambda_{pf}(|\mathbf{W}_2 \odot \mathbf{A}_2|) \lambda_{pf}(|\mathbf{W}_a|) \leq \lambda_{pf}(|\mathbf{W}_2|) \lambda_{pf}(\mathbf{A}_2) \lambda_{pf}(|\mathbf{W}_a|)$. One strict sufficient condition would be:

$$\lambda_{pf}(|\mathbf{W}_2|) \lambda_{pf}(|\mathbf{W}_a|) < \lambda_{pf}(\mathbf{A}_2)^{-1} \quad (4.20)$$

Note that $0 \leq \lambda_{pf}(|\mathbf{W}_2|) < \|\mathbf{W}_2\|_\infty$, and $0 \leq \lambda_{pf}(|\mathbf{W}_a|) < \|\mathbf{W}_a\|_\infty$. When the condition $\|\mathbf{W}_2\|_\infty < 1$ and $\|\mathbf{W}_a\|_\infty < \lambda_{pf}(\mathbf{A}_2)^{-1}$ are satisfied, we will have $\|\mathbf{W}_2\|_\infty \|\mathbf{W}_a\|_\infty < \lambda_{pf}(\mathbf{A}_2)^{-1}$. So Eq. (4.20) will also hold. Similar proof can be established for Eq. (4.28). QED.

The two categories of the instantiations can be naturally incorporated with the downstream tasks which take the input of the solution of the equilibrium equation. The training of the downstream tasks and the equilibrium equation can be conducted in an end-to-end fashion by back propagation. We point out that this could be a future research direction which is closely related to the proposed SYMGNN model. The theoretical properties of the multi-network equilibrium equation can guarantee the existence, uniqueness of the solution, and the convergence of the model training.

4.2.4 The Unified Framework for (Neural) Sylvester Equation Models

Given two input networks with node features $\mathcal{G}_1 = \{\mathbf{A}_1, \mathbf{F}_1\}$, $\mathcal{G}_2 = \{\mathbf{A}_2, \mathbf{F}_2\}$, and the partially observed multi-network association \mathbf{H} of the nodes in \mathcal{G}_1 and \mathcal{G}_2 . First of all, we do not consider the objectives of any downstream tasks, and the goal is to infer all the unobserved entries in \mathbf{H} . Then a unified optimization framework can be written as follows.

$$\mathcal{L}(\mathbf{X}) = \phi(\alpha \cdot a(\mathbf{X}, \mathbf{A}_1, \mathbf{A}_2, \mathbf{F}_1, \mathbf{F}_2) + (1 - \alpha) \cdot b(\mathbf{X}, \mathbf{H}, \mathbf{F}_1, \mathbf{F}_2)) \quad (4.21)$$

where $a()$ and $b()$ are topology smoothing term and anchor multi-network association regularization term, which are similar to Eq. (4.4) and Eq. (4.14). $\phi()$ could be a linear or non-linear function, based on the specific constraints of the optimization problem. α is a weighting scalar for balancing the two terms. The optimization goal of the first term is to strengthen the association of cross-network node pairs if they share similar or consistent local topology. The optimization goal of the second term is to regularize the solution so that the observed multi-network association is consistent with the solution of the optimization problem. For example, if we instantiate Eq. (4.21) by the topology consistency principles proposed in the network alignment problem [30], the framework can be written as:

$$\min_{\mathbf{X}} \mathcal{L}(\mathbf{X}) = \alpha \sum_{a,b,c,d} [\tilde{\mathbf{X}}(a,c) - \tilde{\mathbf{X}}(b,d)]^2 \cdot \mathbf{A}_1(a,b)\mathbf{A}_2(c,d) + (1-\alpha)[\mathbf{X}(a,c) - \mathbf{H}(a,c)]^2 \quad (4.22)$$

Here, for conciseness, we only use topology consistency as an example and do not consider node features. $\tilde{\mathbf{X}}$ is the normalized solution matrix \mathbf{X} by node degrees. (a,b) and (c,d) are two node pairs in \mathcal{G}_1 and \mathcal{G}_2 . In order to solve Eq. (4.22), we can show that it is a convex problem, and the closed-form solution can be obtained by solving a Sylvester equation in the form of Eq. (3.1) presented in Chapter 3 [15, 30]. We can also see that in Eq. (4.22), the $\phi()$ is instantiated as a linear identity function. If we constrain $\phi()$ function as a differentiable non-linear function, the Eq. (4.21) can be seen as a generalization of the multi-network equilibrium equation discussed in Eq. (4.14), since the node features are pruned in the first terms and the solution matrix is removed from the second term in Eq. (4.14).

If we still constrain $\phi()$ function as a differentiable non-linear function, and furthermore, we do not incorporate the solution matrix \mathbf{X} into the first and the second term of Eq. (4.21) to make it in the form of a matrix equation, then we could transform Eq. (4.21) as a neural layer of a feed-forward network as presented in Eq. (4.4). Consequently, the neural layer can be applied to any downstream multi-network mining tasks, and they can be trained through a single loss function in an end-to-end fashion via back propagation [27]. Therefore, Eq. (4.21) could be a unified perspective of the (neural) Sylvester equation-based model discussed in Chapter 3 and Chapter 4. There might exist other potential instantiations of this framework for different optimization goals.

4.3 NOVEL APPLICATION: SOCIAL RECOMMENDATION WITH GNNS

Graph Neural Networks (GNNs) are power tools for a variety of machine learning and data mining tasks on graph data, such as node/graph classification [148] [26], link predic-

tion [149], graph matching [150], and graph-based recommendation [97]. Particularly, social relations play an important role in influencing users’ preference in recommender systems. When choosing from numerous items, users tend to follow their social network friends with whom they share the same interest or whom they trust. This type of trust/interest influence could naturally be captured by GNN models. As a result, substantial recent works focus on applying GNN techniques on the social recommendation task, in which the GNN model is leveraged for learning representations of users and items in a convolutional manner on social networks. Despite some previous successes, the existing works exhibit a few fundamental limitations. First, similar to traditional recommender systems, the negative sampling method for social recommendation is underexplored. One of the most common strategy is uniform negative sampling (UNS), which samples from the unobserved items as negative samples with equal probability. However, the naive UNS method could introduce bias to the model since the unobserved items might also contain positive items. Second, the existing GNN-based neural models usually directly adopt the off-the-shelf GNN models/layers, which often suffer from oversmoothing especially for the social recommendation task. Furthermore, many current GNN architectures are tailored for specific tasks, and might not be suitable for social recommendation if applied directly. Third, the existing social recommendation models do not fully utilize the user-item interaction for message aggregation and representation learning. Most existing methods only consider the observed user-item interactions for users’ interest diffusion for representation learning. However, the unobserved interactions and the negative samples might also provide a different category of interactions, which is often underexplored.

In this section, we design a simplified multi-network GNN-based neural model (NEMOS) for social recommendation problem. Compared with the existing methods, the developed model bears the following distinctive advantages. First, we design a novel generative negative sampling method, which aims at generating hard negative samples as a complement of the sampled negative samples to improve the generalization ability of the model. Second, we develop a simplified multi-network GNN-based model, which does not adopt current GNN models in the user representation learning, but instead only keeps a relative small number of the linear feature aggregation process. By this simplified design, the model could still aggregate hidden features in a convolutional way, and in the meanwhile avoid the oversmoothing issue which is often incurred by the deep Graph Neural Networks. Third, we leverage both the positive and negative user-item interactions in the users’ interest propagation between users and items, in order to explicitly model the users’ positive and negative preferences.

First, we formally define the graph-based social recommendation problem and provide some preliminaries.

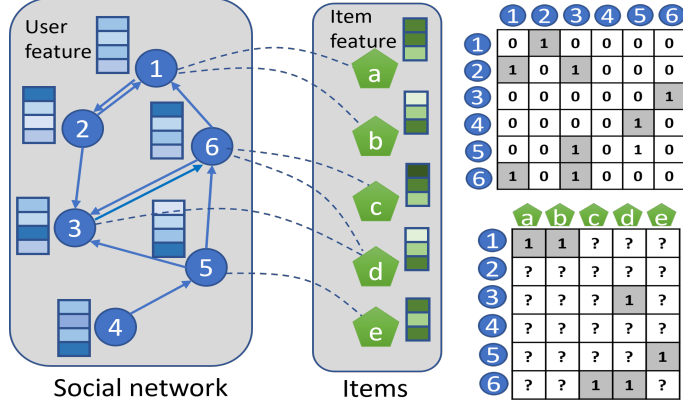


Figure 4.4: Social recommendation problem setting.

4.3.1 Problem Definition

In the traditional recommender system with users' implicit feedbacks, given a user set $\mathcal{U} = \{u_1, u_2, \dots, u_N\}$ and item set $\mathcal{I} = \{i_1, i_2, \dots, i_M\}$, there are only user-item interactions, which could be represented as a matrix $\mathbf{R} \in \mathbb{R}^{N \times M}$. $\mathbf{R}_{ui} = 1$ if there is an observed interaction between user u and item i , and otherwise 0. In the general social recommender system, there is also a social network, whose user-user interactions indicate the users' social relations, and the numerical features of users and items. The social relations can naturally be represented as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We denote the user and item feature matrices as $\mathbf{E} \in \mathbb{R}^{N \times K_1}$ and $\mathbf{F} \in \mathbb{R}^{M \times K_2}$. A toy example of a typical social recommendation setting is illustrated in Figure. 4.4. In real-world recommendation systems, the users could hardly interact with all the items, hence the \mathbf{R} matrix is usually very sparse. Here, its entries with question marks are unobserved. With these inputs, the problem of social recommendation is formally defined as follows.

Definition 4.1. Social Recommendation:

Given: User set \mathcal{U} , item set \mathcal{I} , a social network $G = \{\mathbf{A}, \mathbf{E}, \mathbf{F}\}$, and the observed user-item interaction \mathbf{R} ;

Output: The prediction of the interaction scores of the unobserved user-item pairs for users in \mathcal{U} .

4.3.2 Preliminaries

Classic Matrix Factorization for Social Recommendation. The matrix factorization (MF) is originally used on traditional recommender systems where no social relations are

available. With social information, the MF methods are extended by using the first-order social relation as regularization [151]. A general formulation could be represented as follows.

$$\mathcal{L}_s = \frac{1}{2} \sum_{u=1}^N \sum_{i=1}^M I_{ui} (\mathbf{R}(u, i) - \mathbf{u}_i \mathbf{v}_j^T)^2 + \frac{\beta}{2} \sum_{u=1}^N h(\mathbf{u}_i, \{\mathbf{u}_j\}_{j \in \mathcal{N}_i}) + \frac{\lambda_1}{2} \|\mathbf{U}\|_F^2 + \frac{\lambda_2}{2} \|\mathbf{V}\|_F^2 \quad (4.23)$$

where the I_{ui} is a indicator function to indicate whether (u, i) exists observed interaction. $\mathbf{u}_i, \mathbf{v}_j$ are row vectors of \mathbf{U} and \mathbf{V} . $h(\mathbf{u}_i, \{\mathbf{u}_j\}_{j \in \mathcal{N}_i})$ is the social regularizer function, in which $\{\mathbf{u}_j\}_{j \in \mathcal{N}_i}$ is a set of user i 's neighbors. Representative social regularizer functions are average-based regularization and individual-based regularization, which can be represented as $\|\mathbf{u}_i - \frac{\sum_{j \in \mathcal{N}_i} \mathbf{sim}(i, j) \cdot \mathbf{u}_j}{\sum_{j \in \mathcal{N}_i} \mathbf{sim}(i, j)}\|_F^2$, and $\sum_{j \in \mathcal{N}_i} \mathbf{sim}(i, j) \|\mathbf{u}_i - \mathbf{u}_j\|_F^2$, respectively. The $\mathbf{sim}(i, j)$ denotes a similarity measure between user i and user j .

GNN-based Neural Social Recommendation. As the recent development of Graph Neural Networks (GNN), numerous GNN-based models are proposed for social recommendation. Most existing works focus on how to adopting various GNN models or designing complicate GNN layers for social influence and interest influence diffusion, in order to generate user and item representations. Given the inputs ($G = \{\mathbf{A}, \mathbf{E}, \mathbf{F}\}, \mathbf{R}$), the user and item representation generated by GNN-based neural model at level l can generally be represented as:

$$\mathbf{u}_i^{(l)} = \Phi(\text{GNN}_1(\mathbf{A}, \mathbf{E}), \mathbf{V}^{(l)}) \quad (4.24a)$$

$$\mathbf{v}_j^{(l)} = \text{GNN}_2(\mathbf{R}, \mathbf{E}, \mathbf{F}, \mathbf{U}^{(l)}) \quad (4.24b)$$

where $\text{GNN}_1()$ is used for user representation learning, $\text{GNN}_2()$ is used for item representation learning. $\text{GNN}_2()$'s inputs include \mathbf{R} , which is often treated as an adjacency matrix of user's interest. $\mathbf{U}^{(l)}, \mathbf{V}^{(l)}$ are user and item representations at level l respectively, $\Phi()$ is a neural network for leveraging item representation information back to user representation for learning more compatible user/item representations. As more and more works focus on designing complicate structures for $\text{GNN}_1()$, $\text{GNN}_2()$ and $\Phi()$, our findings suggest that even by a simplified GNN which does not contain multiple sophisticated message-passing layers and nonlinearity, the model can still achieve superior performance.

Next, we present our model with the generative negative sampling strategy, followed by the analysis on complexity.

4.3.3 Model Architecture

We first show the succinct multi-network GNN-based neural model architecture. The key ideas of the proposed model are two-fold. First, the user-user social relations are utilized

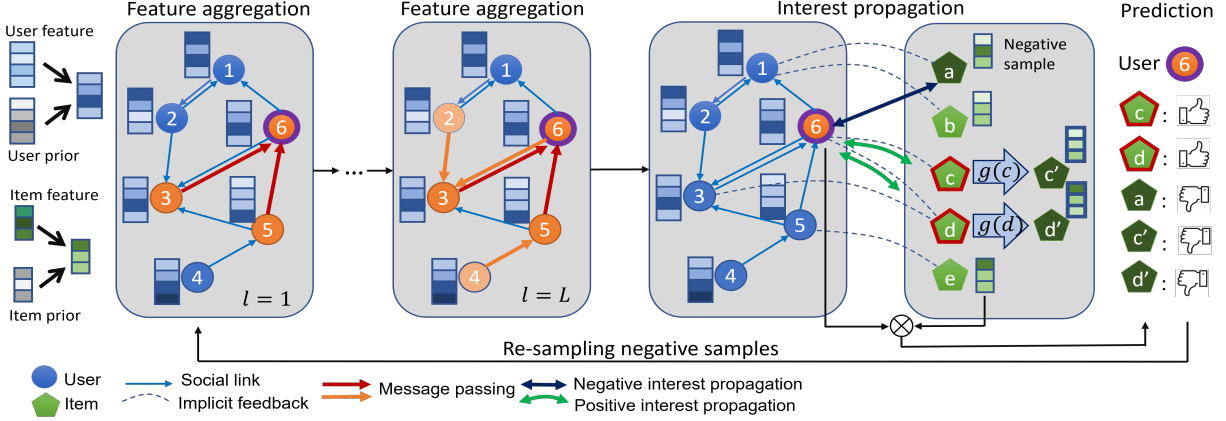


Figure 4.5: The overall model architecture of NEMOS. Best viewed in color.

only for feature aggregation among users, without the complicated architecture designs from various existing GNN models. Here, we hypothesize that the major benefit of the GNN models for social recommendation originates from the local feature aggregation among users in the social network. Furthermore, GNN models often suffer from oversmoothing with multiple layers ([152]). As we will see from the empirical evaluation, users that are multiple hops away provide little or even negative information to the users’ representation learning. Thus the common multi-layer GNN architecture and the nonlinearity between GNN layers are refrained, which also reduces the model complexity. Second, apart from the user-user graph, both the positive and the sampled negative user-item interactions are combined into a heterogeneous bipartite graph, which is used for interest propagation across items and users. The model architecture is illustrated in Figure 4.5.

Figure 4.5 also shows how the input user/item features are processed within one epoch. At the beginning of each epoch, for each user, we uniformly sample a fixed number of items, which do not have interactions with the user. We temporarily treat these items as ‘negative’ items for interest propagation in the model. Note that the ‘negative’ items are not fixed for each epoch. If there is already a set of ‘negative’ samples for each user from the last epoch, a new set of ‘negative’ samples will be re-sampled. A detailed negative sampling strategy is elaborated in Section 4.3.4.

In the model, first of all, the user/item features are fed into a multi-layer perceptron (MLP), and then combined with a user/item prior embedding, which is also known as the ‘free embedding’ in some existing works ([95], [153]), in order to obtain user/item hidden representations for further processing. For example, given user u and item i , the hidden representations can be represented as:

$$\mathbf{H}_1(u, :) = \text{MLP}_1(\mathbf{E}(u, :)) + \mathbf{P}(u, :) \quad (4.25a)$$

$$\mathbf{H}_2(i, :) = \text{MLP}_2(\mathbf{F}(i, :)) + \mathbf{Q}(i, :) \quad (4.25b)$$

where \mathbf{P} and \mathbf{Q} denote the learnable prior representations of users and items, whose initialization $\mathbf{P}(u, :), \mathbf{Q}(i, :) \sim \mathcal{N}(\mathbf{0}, \gamma^2 \mathbf{I})$ follows a Gaussian distribution with zero mean and standard deviation $\gamma > 0$. The intuition of imposing a prior with Gaussian distribution originates from the classic probabilistic matrix factorization for traditional recommender systems. [154] shows that with Gaussian noise as the prior distribution of the latent representations, maximizing the log-posterior over user and item hidden representation is equivalent to minimizing a squared error objective with quadratic regularization. The prior in Eq. (4.25a) (4.25b) work in a similar fashion (see analysis in Section 4.3.7).

Second, user hidden representation is used for the L -level feature aggregation within the social network. In each level, the hidden representations are passed from a source node to a target node of each edge, and the representations are summed up as the updated user hidden representation (\mathbf{H}_1 and \mathbf{H}_2). Third, the resulting user representations and the item representations are diffused in two directions by users' interactions with both positive and the sampled negative items, for positive and negative interest propagation. Here, the users and positive/negative items form a heterogeneous bipartite graph G_b . We use two different MLPs for modeling the positive and negative interest propagation. Then, the positive items are fed into a neural module for generating negative samples. Suppose that $\mathbf{A}_b^+ \in \mathbb{R}^{N \times M}$ denotes the adjacency matrix induced from positive interactions of G_b . $\mathbf{A}_b^+(i, j) = 1$ if user i has interactions with item j . $\mathbf{A}_b^- \in \mathbb{R}^{N \times M}$ denotes the adjacency matrix induced from sampled negative interactions of G_b . $\mathbf{A}_b^-(i, j) = 1$ if user i and item j are sampled as negative pairs. Generally, the intuition behind this module is to generate compatible user (item) embeddings with the assist of the information from item (user). The user embedding $\hat{\mathbf{H}}_1$ and item embedding $\hat{\mathbf{H}}_2$ after interest propagation are:

$$\hat{\mathbf{H}}_1 = \mathbf{A}_b^+ \cdot \mathbf{H}_2 \cdot (\mathbf{D}_u^+)^{-1} + \mathbf{H}_1 \quad (4.26)$$

$$\hat{\mathbf{H}}_2 = \text{MLP}_p((\mathbf{A}_b^+)^T \cdot \mathbf{H}_1 \cdot (\mathbf{D}_i^+)^{-1}) + \text{MLP}_n((\mathbf{A}_b^-)^T \cdot \mathbf{H}_1 \cdot (\mathbf{D}_i^-)^{-1}) + \mathbf{H}_2 \quad (4.27)$$

where $\mathbf{D}_u^+, \mathbf{D}_i^+, \mathbf{D}_i^-$ are diagonal degree matrices for representation normalization. $\mathbf{D}_u^+(x, x) = \sum_y \mathbf{A}_b^+(x, y)$, $\mathbf{D}_i^+(x, x) = \sum_x \mathbf{A}_b^+(x, y)$ and $\mathbf{D}_i^-(x, x) = \sum_y \mathbf{A}_b^-(x, y)$. $\text{MLP}_p()$ and $\text{MLP}_n()$ are two MLP modules for the users' positive and negative interest propagation in item representation respectively. Finally, after the interest propagation, the representations of users and items are used for calculating the final predictions by inner product. The details for loss

function is discussed in Section 4.3.6.

4.3.4 Generative Negative Sampling

Here, we present the proposed generative negative sampling strategy for the social recommendation problem.

Numerous existing works adopt uniform negative sampling (UNS) from the unobserved items. However, it is unreasonable to naively assume that the unobserved items are equal to negative samples, because it might introduce bias into the negative samples, and the UNS method is not able to generate hard negative examples. In order to alleviate this issue, inspired by recent studies on mixup ([155], [156]), which linearly interpolates pairs of examples for data augmentation in various tasks, we introduce the generative negative sampling. Our key idea is to generate negative samples through neural networks in the continuous embedding space of items. For a given user u , the neural generator takes the hidden representations of true positive items \mathbf{z}_i as inputs, and the representations of uniformly sampled unobserved items $\{i_1, \dots, i_p\}$ as offsets:

$$\mathbf{z}'_i = \alpha \cdot g(\mathbf{z}_i) + (1 - \alpha) \cdot \text{Pooling}_{i_1, \dots, i_p \in \Omega_-}(\mathbf{z}_{i_1}, \dots, \mathbf{z}_{i_p}) \quad (4.28)$$

where $g(\mathbf{z}_i) = \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \cdot \mathbf{z}_i + \mathbf{b}_1) + \mathbf{b}_2)$ is a MLP, with $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ as learnable parameters, and $\sigma()$ is a Sigmoid function. Ω_- is the set of sampled unobserved items. $\text{Pooling}()$ is the Pooling function which transforms the input representations of the sampled unobserved items into a combined representation. α is a weighting scalar for the first MLP term and the second offset term of unobserved samples. The intuition of the first MLP term is to deviate the true item representation from the embedding space in order to generate a *fake* item representation, whose corresponding item might not exist. The intuition of the second term is to bump such deviation in the direction of unobserved samples, in which the true negative samples might exist. The above negative sample generator is learned with the model in an end-to-end fashion. Note that the generated samples may take different set of sampled negative items as inputs for Eq. (4.28), so that the embeddings of the generated negative samples are also learned along with the learning process of the recommendation problem. As the learning process proceeds, the generated negative could become harder, which is closely related to the idea of curriculum learning [157, 158]. We will further discuss this in Section 4.3.8.

There are two major advantages by the generative negative sampling. First, the generated samples are mixed with information from positive items, resulting in harder negative samples

compared with samples by UNS. Second, the generated fake samples can be combined with the real samples as an augmentation for the dataset. In this way, the proposed model is potentially more generalizable because of the interpolation of embedding spaces with augmentation during training.

4.3.5 Implementation Details

Here, we further present some implementation details.

Choice of α . As suggested by [155], the selection of α affects the model generalization, and it is often solved by sampling from a distribution. Here we use Beta distribution due to its strong empirical performance.

Static Negative Re-sampling. As a natural extension for UNS, we can uniformly re-sample the negative samples from unobserved items in every epoch of the training, instead of using one fixed negative item set. Static means the sampling method does not consider the model output dynamically as in dynamic negative sampling. Compared with fixing the negative item set, this could potentially prevent overfitting and also improve the model’s generalization.

Dynamic Negative Re-sampling. We also adopt dynamic negative sampling for mitigating the limitations of UNS and obtaining hard negative samples dynamically at each epoch. Specifically, the unobserved items are ranked by the model’s output rating scores at each training epoch. Then the negative samples are extracted from the top ranked items, which are supposed to be the hard examples.

4.3.6 Training

Instead of leveraging the commonly used Bayesian Personalized Ranking (BPR) loss, we adopt the simple Mean Squared Error (MSE) loss, which is easier to implement for our model and meanwhile with superior performance. We briefly discuss two limitations of the BPR loss in terms of practical implementation. First, given a pair of positive and negative samples, BPR loss ranks the positive sample higher than the negative sample. If the pair is uniformly sampled from the observed and unobserved items, it always makes the ratio of positive and negative items equal to 1 : 1, which restricts the model generalization. Second, for C positive items, we sample rC negative samples using negative sampling ratio r , and consider every combination of positive and negative item pairs in BPR loss. The computational complexity is $O(N \cdot rC^2)$, which is significantly larger than MSE loss ($O(N \cdot (C + rC))$). Our loss

function is given as follows.

$$\mathcal{L} = \sum_{(u,i) \in \Omega_+ \cup \Omega_-} \|\mathbf{R}(u, i) - \hat{\mathbf{H}}_1(u, :) \hat{\mathbf{H}}_2(i, :)^{\top}\|_2^2 \quad (4.29)$$

The predicted rating of (u, i) is calculated as the inner product of representation vectors $\hat{\mathbf{H}}_1(u, :) \hat{\mathbf{H}}_2(i, :)^{\top}$. Ω_+ and Ω_- are positive and negative user-item pair set respectively. Here, for Ω_- , we combine the uniformly re-sampled items with the generated fake items as a given user’s final negative samples to obtain the best performance. We adopt the Adam optimizer which shows more stable convergence than other optimizers. The Adam optimizer is applied with a weight decay of 0.001.

4.3.7 Analysis

In this subsection, we first discuss the effect of the Gaussian prior, and then give the complexity analysis for NEMOS.

Gaussian Prior. Here, under the same assumption as probabilistic matrix factorization, the conditional probability of the observed user-item interaction follows a Gaussian distribution:

$$p(\mathbf{R}|\mathbf{P}, \mathbf{Q}, \gamma_R^2) = \prod_{u=1}^N \prod_{i=1}^M [\mathcal{N}(\mathbf{R}(u, i) | f(\mathbf{p}_u) f(\mathbf{q}_i)^{\top}, \gamma_r^2)]^{I_{u,i}^{\mathbf{R}}} \quad (4.30)$$

where \mathbf{p}_u and \mathbf{q}_i are the u -th row vector and the i -th row vector in \mathbf{P}, \mathbf{Q} respectively. $f()$ function represents the model, and here we assume that the rest of the parameters are fixed. $I_{u,i}^{\mathbf{R}}$ is an indicator function, which is equal to 1 if (u, i) has observed interactions, and otherwise 0. Given that $\mathbf{P}(u, :), \mathbf{Q}(i, :) \sim \mathcal{N}(\mathbf{0}, \gamma^2 \mathbf{I})$, the posterior probability of the \mathbf{Q} and \mathbf{P} is:

$$p(\mathbf{P}, \mathbf{Q} | \mathbf{R}, \mathbf{A}, \gamma^2, \gamma_R^2) \propto p(\mathbf{R} | \mathbf{P}, \mathbf{Q}, \gamma_R^2) p(\mathbf{P} | \gamma^2) p(\mathbf{Q} | \gamma^2) \quad (4.31)$$

We plug Eq. (4.30), $p(\mathbf{P} | \gamma^2)$, and $p(\mathbf{Q} | \gamma^2)$ into Eq. (4.31), and maximize the log-posterior becomes equal to minimizing the following formula:

$$\mathcal{L}' = \frac{1}{2} \sum_{u=1}^N \sum_{i=1}^M I_{u,i}^{\mathbf{R}} (\mathbf{R}(u, i) - f(\mathbf{p}_u) f(\mathbf{q}_i)^{\top})^2 + \frac{\gamma_R^2}{2\gamma^2} \sum_{u=1}^N \|\mathbf{P}(u, :)\|_2^2 + \frac{\gamma_R^2}{2\gamma^2} \sum_{i=1}^M \|\mathbf{Q}(i, :)\|_2^2 \quad (4.32)$$

We can see that the last two terms in Eq. (4.32) indicate the L2-regularization of the learnable prior representations \mathbf{P}, \mathbf{Q} .

Complexity Analysis. The major computational hurdle lies in the user feature aggregation and the interest propagation. Suppose that the adjacency matrix of user-user social network

contains m_1 non-zero entries, and the adjacency matrix of the bipartite user-item graph contains m_2 and m_3 non-zero entries for $\mathbf{A}_b^+, \mathbf{A}_b^-$ respectively. Let the dimension of the user/item hidden representations equal to d . Then the time complexity of the proposed model is $O(\text{iter} \cdot (K_1 m_1 + d(m_2 + m_3)))$, where iter indicates the number of iterations in all forward pass. Since the adjacency matrix \mathbf{A} , \mathbf{A}_b^+ , and \mathbf{A}_b^- are all usually sparse in real-world data, m_1, m_2, m_3 are usually comparable or smaller than the magnitude of N, M , and are much smaller when compared with the magnitude of N^2 and M^2 .

4.3.8 Discussion

Here, we discuss the relation between the proposed generative negative sampling strategy with the curriculum learning method on graphs, particularly in recent research for node and graph classification. The idea of curriculum learning adopts the idea of training a machine learning model in an easy-to-difficult fashion. For graph classification, *CurGraph* by Wang et al. [158] proposes to calculate a difficulty score for the graphs in training data via the graph and node embeddings of the Graph Neural Network model. *GNN-CL* by Li et al. [157] proposes a node generator and edge generator empowered GNN model for imbalanced node classification problem. Specifically, the node generator uses the node embeddings from the minority class as inputs for generating new node embeddings as a oversampling strategy for minority class. This method can be seen as a special case of our proposed generative negative sampling method, when the embeddings of the positive items are not used as inputs for the negative sample generation.

Lastly, we present the experimental results on real-world datasets to show the effectiveness of the proposed model.

4.3.9 Experimental Setting

We use the most widely used two benchmark datasets in the experiments, and their statistics are shown in Table. 4.6.

Table 4.6: The statistics of the two benchmark datasets.

Dataset	Yelp	Flickr
# of users	17,237	8,358
# of items	38,342	82,120
# of ratings	204,448	143,765
# of observed links	0.03%	0.05%
Link density	0.05%	0.27%

Datasets and Pre-processing. *Yelp.com* is an online website that publishes crowd-sourced reviews and ratings about businesses. Users can also connect to each other to have a social relation through *Yelp*. The ratings are in the range of $[0, 5]$, and the reviews are usually text and images. The *Yelp* dataset² mainly consists of information about users, businesses, and reviews. For the pre-processing, we use the same process setting as the baselines [94], [95]. The user-user social network is constructed via the user’s friend information. The user/item feature vectors are calculated by averaging all the learned word embeddings of the user/item via Word2vec model.

Flickr is an American image hosting and video hosting service, as well as an online community. Users can follow each other and build social relations. The items (photos and videos) can be upvoted by users as implicit feedbacks. The *Flickr* dataset is shared by [94]. For the pre-processing, the user feature vectors are calculated by averaging the image feature representations he/she liked generated via a VGG16 convolutional neural network.

Baseline Methods. We compare the NEMOS model with three types of baseline methods: (1) one representative traditional recommendation method without the usage of social relations (*BPR* [159]); (2) one representative traditional social recommendation methods which models the first-order social relations (*CNSR* [91]); and (3) six state-of-the-art GNN-based social recommendation methods (*GraphRec* [44], *PinSage* [92], *NGCF* [93], *DiffNet* [94], *DiffNet++* [95], *DiffNetLG* [96]).

Experimental Settings. For the metrics, we use two most commonly used metrics for recommendation, Hit Ratio ($HR@K$) and Normalized Discounted Cumulative Gain ($NDCG@K$), where $K \in \{5, 10, 15\}$. In the experiment, the training, validation and testing data ratio is equal to $8 : 1 : 1$, and we use the exact same split for all baselines. In the evaluation, we sample 1,000 unrated items for each user, and combine them with the rated items for the calculation of $HR@K$ and $NDCG@K$. The results are averaged over five runs.

Hyperparameter Settings. For *Yelp* dataset, the number of uniformly sampled unobserved items for each user is 8. For *Flickr* dataset, the number of uniformly sampled unobserved items for each user is 15. For both datasets, we use 2 layers of user feature aggregation, 200 as batch size, and 0.001 as learning rate, as we find them contributing to the best evaluation performance.

4.3.10 Effectiveness Results

The performance comparison is shown in Table 4.7 and Table 4.8. ‘Stat.’ means training with static re-sampling strategy, and ‘Gen.’ indicates training with generative negative

²<https://www.yelp.com/dataset>

Table 4.7: Social recommendation comparison on *Yelp* dataset.

Models	HR@K			NDCG@K		
	K=5	K=10	K=15	K=5	K=10	K=15
BPR	0.1695	0.2632	0.3252	0.1231	0.1554	0.1758
CNSR	0.1877	0.2904	0.3458	0.1389	0.1746	0.1912
GraphRec	0.1915	0.2912	0.3623	0.1279	0.1812	0.1956
PinSage	0.2105	0.3049	0.3863	0.1539	0.1828	0.2130
NGCF	0.1992	0.3042	0.3863	0.1450	0.1828	0.2041
DiffNet	0.2276	0.3461	0.4217	0.1679	0.2118	0.2307
DiffNet++	0.2503	0.3694	0.4493	0.1841	0.2263	0.2497
DiffNetLG	0.2599	0.3711	<u>0.4473</u>	0.1941	0.2333	0.2586
Ours (Stat.)	<u>0.3857</u>	<u>0.3918</u>	0.3982	<u>0.4093</u>	<u>0.4105</u>	<u>0.4122</u>
Ours (Gen.)	0.3956	0.4025	0.4146	0.4198	0.4212	0.4231

Table 4.8: Social recommendation comparison on *Flickr* dataset.

Models	HR@K			NDCG@K		
	K=5	K=10	K=15	K=5	K=10	K=15
BPR	0.0651	0.0795	0.1037	0.0603	0.0628	0.0732
CNSR	0.0920	0.1229	0.1445	0.0791	0.0978	0.1057
GraphRec	0.0931	0.1231	0.1482	0.0784	0.0930	0.0992
PinSage	0.0934	0.1257	0.1502	0.0844	0.0998	0.1046
NGCF	0.0891	0.1189	0.1399	0.0819	0.0945	0.0998
DiffNet	0.1178	0.1657	0.1855	0.1072	0.1271	0.1301
DiffNet++	0.1412	0.1832	0.2203	0.1296	0.1420	0.1544
Ours (Stat.)	<u>0.1452</u>	0.1411	0.1412	<u>0.1725</u>	<u>0.1674</u>	<u>0.1667</u>
Ours (Gen.)	0.1547	<u>0.1504</u>	<u>0.1469</u>	0.1850	0.1792	0.1781

sampling strategy. The best performances are shown in bold font and the second best performances are shown with underlines. On *Yelp* dataset, except for HR@15, our NEMOS model significantly outperforms all baseline methods. Specifically, the HR@5 is improved by 34.3% compared with the best baseline. The NDCG@K is also significantly improved. For example, the NDCG@15 is increased by up to 38.8%. On *Flickr* dataset, similar observations can be made. The HR@5 is increased by 8.7% and the NDCG@15 is increased by 13.3% compared with the best baseline. Furthermore, the proposed model shows more significant improvement when the *K* is small.

4.3.11 Ablation Study

The ablation study results are shown in Table 4.9 and Table 4.10. The 'Uni. prior' represents the model trained with a uniformly distributed prior user/item representations. The

Table 4.9: Results for ablation study on *Yelp* dataset.

Models	HR@K			NDCG@K		
	K=5	K=10	K=15	K=5	K=10	K=15
Uni. prior	0.2718	0.2747	0.2790	0.2935	0.2937	0.2948
No prior	0.1591	0.1617	0.1654	0.1698	0.1703	0.1714
GCN	0.3541	0.3532	0.3533	0.3777	0.3767	0.3766
No re-sample	0.3713	0.3764	0.3821	0.3950	0.3967	0.3983
Ours (Stat.)	<u>0.3857</u>	<u>0.3918</u>	<u>0.3982</u>	<u>0.4093</u>	<u>0.4105</u>	<u>0.4122</u>
Ours (Gen.)	0.3956	0.4025	0.4146	0.4198	0.4212	0.4231

'No prior' represents training without the prior user/item representation. 'GCN' denotes our model variant in which the user feature aggregation is replaced with a Graph Convolution Network. 'No re-sample' denotes training without the negative re-sampling process at each epoch. As we can see from the results, firstly the prior representation has a huge impact on the model performance. Without such prior, the performance could drop up to 59.8% for HR. Initializing the prior with proper distribution is also important, since Gaussian distribution outperforms the uniform distribution. As we discuss in the Gaussian prior, the Gaussian distribution regularizes the learnable prior representations. Secondly, using GNN model does not positively contribute to the final performance. As we discuss in the model, the non-linearity and complex design in GNN models might have little or even negative influence. Instead, a simple two-layer linear user feature aggregation performs the best. Thirdly, the re-sampling strategy is crucial and could improve the model's ability of generalization. Generally, the generative negative sampling outperforms the static negative sampling and further outperforms no re-sampling.

Table 4.10: Results for ablation study on *Flickr* dataset.

Models	HR@K			NDCG@K		
	K=5	K=10	K=15	K=5	K=10	K=15
Uni. prior	<u>0.1479</u>	<u>0.1442</u>	<u>0.1443</u>	<u>0.1758</u>	<u>0.1706</u>	<u>0.1700</u>
No prior	0.0840	0.0812	0.0808	0.0991	0.0957	0.0951
GCN	0.0687	0.0890	0.1125	0.0693	0.0687	0.0721
No re-sample	0.1045	0.1022	0.1020	0.1213	0.1186	0.1182
Ours (Stat.)	0.1452	0.1411	0.1412	0.1725	0.1674	0.1667
Ours (Gen.)	0.1547	0.1504	0.1469	0.1850	0.1792	0.1781

As discussed in previously, we show the impact of aggregation layers for user features in Figure 4.6. Both the proposed model and the model using GCN module are tested. As we can see, for both model variants, the performance reaches the peak at 2 layers of aggregation,

and quickly drops at larger number of layers. It also suggests that it might not be necessary to use deep GNN models in many social recommendation tasks as the representation becomes oversmoothing fast.

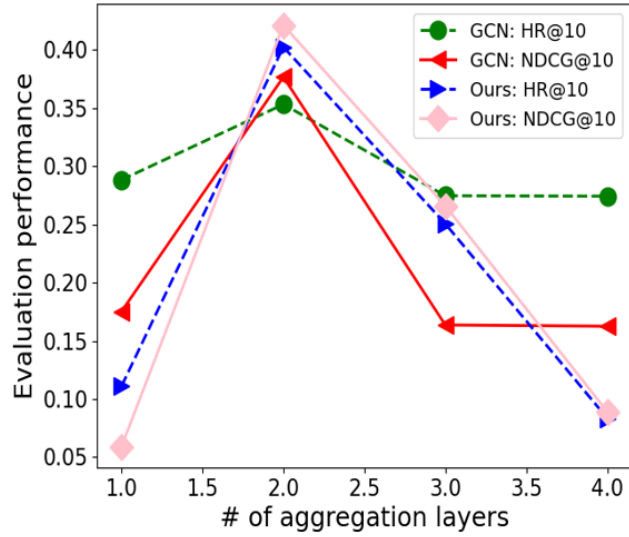


Figure 4.6: The impact of aggregation layers.

CHAPTER 5: HIGH-ORDER ASSOCIATION WITH NUMERICAL TECHNIQUES

In this chapter, we will elucidate our works for the problem of multi-way association for both plain networks and attributed networks [15], which belongs to the category of high-order multi-network association in Table 1.1. Compared with the pair-wise association, much fewer methods exist for multi-way association due to the challenges in problem formulation, algorithm design, and the scalability issue. We will show how to model the problem with a convex optimization formulation, and how to solve it by efficient approaches.

5.1 BACKGROUND AND MOTIVATION FOR MULTI-WAY ASSOCIATION

Multiple networks are ubiquitous in many important applications. For example, how to link identical or similar users from multiple social networks (i.e., collective network alignment) [39]? How to simultaneously recommend items, activities as well as locations to a user (i.e., high-order recommendation) [34, 71]? In bioinformatics, how to discover relevant drugs and genes for a specific disease [160]? In team management, how to optimally assign team members with the right skills to the right teams for relevant tasks [132, 161]? The key to answering these questions lies in *multi-way association*, which identifies strongly correlated nodes from multiple networks.

For pair-wise association (e.g., network alignment [18, 30, 81, 162]), it links *node-pairs* across two input networks. On the contrary, multi-way association aims to discover the collective association w.r.t. to *a set of nodes*. Figure 5.1 presents an illustrative example. Given three social networks, a multi-way association (e.g., the red dashed box on the left of Figure 5.1) is a set of three users (i.e. nodes) from each of the three input social networks, who are either identical or similar with each other. We can represent all the multi-way associations in the form of a 3rd-order tensor \mathcal{X} (e.g., the right-most part in Figure 5.1), and the weights of the tensor entries measure the strength of the corresponding multi-way associations. For a given user, each entry of the corresponding *slice* of tensor \mathcal{X} with a high weight indicates *a pair of* users from the other two social networks who are both strongly associated with the given user. Likewise, given an activity network, a social network and a location network, strong multi-way associations inferred from them could indicate that the corresponding activities, users, and locations are associated with each other.

Compared with the pair-wise association, much fewer methods exist for multi-way association due to a number of challenges. First (*C1. Problem Formulation*), pair-wise cross-network association is often formulated as an optimization problem [30, 81, 162]. For exam-

ple, soft network alignment [30, 162, 163] can be formulated as a convex optimization problem based on the alignment consistency principle as we have shown in Chapter 3. However, it is not clear if such a consistency principle would generalize to multi-way association. Second (*C2. Algorithms*), even if we can formulate multi-way association from the optimization perspective, it is still highly challenging to solve it in terms of its optimality and sensitivity. Third (*C3. Scalability*), the solution space of multi-way association is significantly larger than pair-wise association. To see this, suppose there are K input networks, each with n nodes. There could be as many as n^K multi-way associations. Therefore, another major hurdle is to scale-up the multi-way association algorithm to large networks.

In this chapter, we address these three challenges, and our main contributions are summarized as follows.

- **Formulation.** We formulate the multi-way association problem as an optimization problem and show that it can be solved optimally by a Sylvester tensor equation.
- **New Algorithms.** We propose two fast algorithms (SYTE-Fast-P and SYTE-Fast-A) to solve the Sylvester tensor equation on plain networks and attributed networks respectively, with a linear complexity in both time and space.
- **Proofs and Analysis.** We provide theoretic analysis of the proposed algorithms in terms of optimality, sensitivity and complexity.
- **Empirical Evaluations.** We conduct extensive empirical evaluations on a diverse set of real networks which demonstrate the efficacy of the proposed methods.

5.2 MULTI-WAY ASSOCIATION ON PLAIN NETWORKS

In this section, we focus on the multi-way association problem on plain networks. We first formally give the problem definition of the multi-way association inference, and then introduce a convex optimization formulation for solving it. Next, we elaborate a basic algorithm, which uses the fixed point iteration method, and two Krylov subspace-based algorithms. The algorithms are followed by analysis on the complexity.

5.2.1 Problem Definition

We first briefly introduce the notations used in the paper. We use bold uppercase letters to represent matrices, bold lowercase letters to represent vectors, bold calligraphic letters to

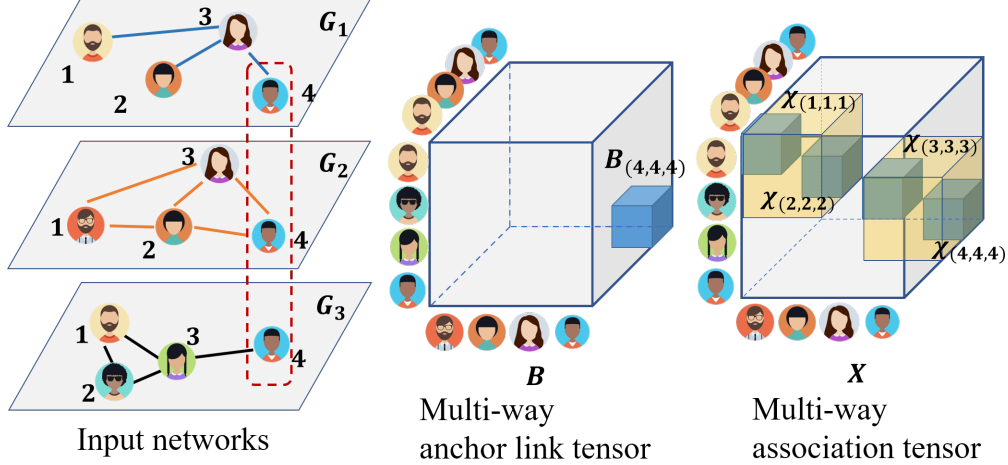


Figure 5.1: An illustrative example of multi-way association for collective social network alignment. Left (from top to bottom): three input social networks, such as LinkedIn, Facebook and Twitter. The red dashed rectangle indicates a prior anchor link (\mathcal{B} in the middle). Right: solution tensor \mathcal{X} of the corresponding Sylvester tensor equation gives the inferred multi-way association; the highlighted cubes in \mathcal{X} denote four strong multi-way associations. Best viewed in color.

represent tensors, lowercase or uppercase letters in regular font for scalars, and calligraphic letters for multi-index (e.g. \mathfrak{L}). Specifically, each network G is represented by three matrices, including the (weighted) adjacency matrix \mathbf{A} , the edge attribute matrix \mathbf{E} and the node attribute matrix \mathbf{N} . The i -th element of a vector \mathbf{x} is denoted as \mathbf{x}_i , the element (i, j) of a matrix \mathbf{A} is denoted as $\mathbf{A}(i, j)$, and the element (i_1, i_2, \dots, i_k) of a tensor \mathcal{X} is denoted as $\mathcal{X}(i_1, i_2, \dots, i_k)$. We focus on categorical edge and node attributes in this paper. \mathbf{E} has the same dimension as its corresponding adjacency matrix \mathbf{A} , and $\mathbf{E}^p(i, j) = 1$ if the edge (i, j) has edge attribute p . \mathbf{N} is a diagonal matrix, and $\mathbf{N}^q(i, i) = 1$ if node i has node attribute q . We use similar notations for tensors as described in [164]. For example, \times_k denotes the k -mode product of tensors, and $\mathcal{X}_{(k)}$ denotes the mode- k unfolding. We define $\text{vec}(\cdot)$ and $\text{reshape}(\cdot, n_1, \dots, n_K)$ as vectorizing operation and tensorizing operator. In general, we need to specify orders of vectorization and tensorization in tensor. By default, we let $\text{vec}(\cdot)$ and $\text{reshape}(\cdot, n_1, \dots, n_K)$ be vectorizing and tensorizing operations along the 1-st mode.¹ For example, $\text{vec}(\mathcal{X}) = \text{vec}(\mathcal{X}_{(1)})$, i.e., column-wise vectorization of the mode-1 unfolding of tensor \mathcal{X} . $\text{reshape}(\mathbf{x}, n_1, \dots, n_K) = \mathcal{X}$ tensorizes vector \mathbf{x} to a tensor of dimension $n_1 \times n_2 \times \dots \times n_K$ by stacking the vectors of length n_1 as the mode-1 fibers of \mathcal{X} . For brevity, we assume that the K input networks share comparable numbers of

¹ $\text{vec}_k(\cdot)$ will be used to denote vectorization along the k -th mode when necessary.

nodes (i.e., $O(n_1) = \dots = O(n_K) = O(n)$) and comparable numbers of edges (i.e., $O(m_1) = \dots = O(m_K) = O(m)$). With these notations, we formally define the multi-way association tensor (Definition 5.1), the multi-way anchor link tensor (Definition 5.2), and the multi-way association inference problem (Problem 5.1) as follows.

Table 5.1: Table of Symbols

Symbols	Definitions and Descriptions
$G = \{\mathbf{A}, \mathbf{E}, \mathbf{N}\}$	An attributed network
$\mathbf{A}, \tilde{\mathbf{A}}$	Original/normalized adjacency matrix
\mathbf{E}	Edge attribute matrix
\mathbf{N}	Node attribute matrix
\mathcal{B}	Multi-Way anchor link tensor
\mathcal{X}	Multi-Way association tensor
\mathbf{x}, \mathbf{b}	Vectorized tensor \mathcal{X} and \mathcal{B}
K	Number of input networks
P, Q	Number of edge/node attributes
n_i, m_i	# of nodes/edges of $G_i (i = 1, \dots, K)$
s	Rank of multi-way anchor link tensor
r	Rank used in eigen-decomposition
$\text{vec}(\cdot)$	Vectorization operator along 1-st mode
$\text{reshape}(\cdot, n_1, \dots, n_K)$	Tensorization operator along 1-st mode
$\bigotimes_{i=1}^K$	Kronecker products from index 1 to K

Definition 5.1. MULTI-WAY ASSOCIATION TENSOR

Given a set of K networks G_k ($k = 1, \dots, K$). A multi-way association is the association of a set of nodes (i_1, i_2, \dots, i_K) , where $i_1 \in G_1, i_2 \in G_2, \dots, i_K \in G_K$. The multi-way association tensor \mathcal{X} is of size $n_K \times n_{K-1} \times \dots \times n_1$. Each entry $\mathcal{X}(i_k, i_{k-1}, \dots, i_1)$ indicates to what extent the corresponding nodes $(i_k, i_{k-1}, \dots, i_1)$ are associated with each other.

Definition 5.2. MULTI-WAY ANCHOR LINK TENSOR

Given a set of K networks G_k ($k = 1, \dots, K$). The multi-way association prior tensor \mathcal{B} is of size $n_K \times n_{K-1} \times \dots \times n_1$. If the corresponding nodes in $(i_K, i_{K-1}, \dots, i_1)$ are known to be associated with each other a priori, we call $(i_K, i_{K-1}, \dots, i_1)$ an anchor link.

For the example in Figure 5.1, only one multi-way anchor link (marked by the red rectangle) is known a priori, and thus there is only one non-zero entry in \mathcal{B} .² Each positive entry in \mathcal{X} represents a multi-way association between a set of three users, one from each of the

²For clarity, we assume that a sparse anchor link tensor \mathcal{B} is given for the rest of this paper. Nonetheless, such an anchor link tensor is optional, and our proposed formulation, algorithms and analysis can be naturally adapted in the absence of \mathcal{B} . For example, if no multi-way association is available a priori, we can set \mathcal{B} as a uniform tensor, i.e. each entry is equal to 1.

three input social networks. By the multi-way association inference algorithms that will be introduced in the subsequent sections, we might find that there are four positive entries in tensor \mathcal{X} (e.g., $\mathcal{X}(1, 1, 1)$, $\mathcal{X}(2, 2, 2)$, $\mathcal{X}(3, 3, 3)$ and $\mathcal{X}(4, 4, 4)$). Each of these four entries indicates a set of three users who are either identical or similar with each other.

Problem 5.1. MULTI-WAY ASSOCIATION INFERENCE

Given: A set of K networks $\{G_k (k = 1, \dots, K)\}$, and a multi-way anchor link tensor \mathcal{B} with dimension $n_K \times n_{K-1} \times \dots \times n_1$;

Output: The multi-way association tensor \mathcal{X} , whose entries measure the strength of the association of the corresponding node sets.

5.2.2 Formulation and Basic Algorithm

SyTE Formulation. We first formulate Problem 5.1 from the optimization perspective. The key idea is to generalize the *consistency* principle which was originally designed for pairwise network alignment [30]. Given two sets of nodes (i_K, \dots, i_1) and (j_K, \dots, j_1) . Intuitively, the consistency principle requires that the multi-way association of (i_K, \dots, i_1) be consistent with that of (j_K, \dots, j_1) (i.e., $\mathcal{X}(i_K, \dots, i_1) \approx \mathcal{X}(j_K, \dots, j_1)$), if the two node sets are consistent in terms of the following three aspects. This includes (1) *topology consistency*, meaning that the two node sets are strongly connected with each other (i.e., large $\mathbf{A}_k(i_k, j_k)$ ($k = 1, \dots, K$)); (2) *node attribute consistency*, meaning that nodes in each of the two sets share the same attributes respectively (i.e., $\mathbf{N}_1(i_1, i_1) = \dots = \mathbf{N}_K(i_K, i_K)$ and $\mathbf{N}_1(j_1, j_1) = \dots = \mathbf{N}_K(j_K, j_K)$); and (3) *edge attribute consistency*, meaning that the two node sets are connected with each other by the same edge attribute (i.e., $\mathbf{E}_1(i_1, j_1) = \dots = \mathbf{E}_K(i_K, j_K)$). Formally, this leads to the following objective function.

$$\begin{aligned}
 J(\mathcal{X}) = & \sum_{\substack{i_1, \dots, i_K \\ j_1, \dots, j_K}} \left[\beta \left(\frac{\mathcal{X}(i_K, \dots, i_1)}{\sqrt{d(i_1, \dots, i_K)}} - \frac{\mathcal{X}(j_K, \dots, j_1)}{\sqrt{d(j_1, \dots, j_K)}} \right)^2 \underbrace{t(\mathbf{A}_1 \dots \mathbf{A}_K)}_{\text{Topology consistency}} \right. \\
 & \times \underbrace{f(i_k) \times f(j_k)}_{\text{Node attribute consistency}} \times \underbrace{g(i_k, j_k)}_{\text{Edge attribute consistency}} \\
 & \left. + \underbrace{(1 - 2\beta)(\mathcal{X}(i_K, \dots, i_1) - \mathcal{B}(i_K, \dots, i_1))^2}_{\text{Anchor link regularizer}} \right] \tag{5.1}
 \end{aligned}$$

where $\beta \in (0, 0.5)$ is a weighting parameter, and functions $f(\cdot)$ and $g(\cdot)$ denote the node and edge attribute consistency terms. The topology consistency term for all networks is $t(\mathbf{A}_1 \dots \mathbf{A}_K) = \mathbf{A}_1(i_1, j_1) \dots \mathbf{A}_K(i_K, j_K)$. Function $d(\cdot)$ denotes node-set normalization. They are defined as $f(i_k) = \mathbb{1}(\mathbf{N}_1(i_1, i_1) = \dots = \mathbf{N}_K(i_K, i_K))$, $g(i_k, j_k) = \mathbb{1}(\mathbf{E}_1(i_1, j_1) =$

$\cdots = \mathbf{E}_K(i_K, j_K)$), where $\mathbf{1}(\cdot)$ is the indicator function. The denominator $d(i_1, \dots, i_K) = \sum_{j_1, \dots, j_K} \mathbf{A}_1(i_1, j_1) \cdots \mathbf{A}_K(i_K, j_K)$, if $f(i_k) = f(j_k) = g(i_k, j_k) = 1$; otherwise $d(i_1, \dots, i_K) = 1$. By using the notation defined in Table 5.1, the consistency terms $f(\cdot)$ and $g(\cdot)$ can be re-written as $f(i_k) = \sum_{p=1}^P \mathbf{N}_1^p(i_1, i_1) \cdots \mathbf{N}_K^p(i_K, i_K)$, and the edge attribute consistency term $g(i_k, j_k) = \sum_{q=1}^Q \mathbf{E}_1^q(i_1, j_1) \cdots \mathbf{E}_K^q(i_K, j_K)$.

By vectorizing the multi-way association tensor, the objective function in Eq. (5.1) can be re-written in a more concise form. First, let the tensor indices i_K, i_{K-1}, \dots, i_1 and j_K, j_{K-1}, \dots, j_1 become vector indices u and v respectively, where $u = \sum_{k=1}^{K-1} \prod_{j=k+1}^K n_j(i_k - 1) + i_K$, and $v = \sum_{k=1}^{K-1} \prod_{j=k+1}^K n_j(j_k - 1) + j_K$. Then, note that $d(\cdot)$ in Eq. (5.1) actually calculates the degree matrix of a Kronecker graph formed by $\mathbf{A}_1, \dots, \mathbf{A}_K$ after filtered by the node and edge attributes. Specifically, let: $\mathbf{W} = \sum_{i,j,k} \mathbf{N}_1^i(\mathbf{E}_1^k \odot \mathbf{A}_1) \mathbf{N}_1^j \otimes \cdots \otimes \mathbf{N}_K^i(\mathbf{E}_K^k \odot \mathbf{A}_K) \mathbf{N}_K^j = \mathbf{N}(\mathbf{E} \odot (\mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_K)) \mathbf{N}$ where $\mathbf{N} = \sum_{i=1}^P \mathbf{N}_1^i \otimes \cdots \otimes \mathbf{N}_K^i$, $\mathbf{E} = \sum_{k=1}^Q \mathbf{E}_1^k \otimes \cdots \otimes \mathbf{E}_K^k$, and \odot is the element-wise product. Let \mathbf{D} be the diagonal degree matrix of \mathbf{W} such that $\mathbf{D}(u, u) = \sum_v \mathbf{W}(u, v)$, and let $\mathbf{x} = \text{vec}(\mathcal{X})$, $\mathbf{b} = \text{vec}(\mathcal{B})$. Plugging all these into Eq. (5.1), we have the following objective function to minimize w.r.t. multi-way association vector \mathbf{x} .

$$\arg \min_{\mathbf{x}} J(\mathbf{x}) = \alpha \mathbf{x}^\top (\mathbf{I} - \widetilde{\mathbf{W}}) \mathbf{x} + (1 - \alpha) \|\mathbf{x} - \mathbf{b}\|_2^2 \quad (5.2)$$

where $\widetilde{\mathbf{W}} = \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$. The first term in Eq. (5.2) is equivalent to the first term of summation in Eq. (5.1) for consistency principles, and the second term is to encode the prior knowledge of anchor links. The weighting parameter is reset as $0 < \alpha < 1$ which balances the consistency objective and prior knowledge of anchor links. It can be shown that $\alpha = 2\beta$. **Basic Algorithm.** We show that Eq. (5.2) is a convex problem in Lemma 5.2.2. Thus, its fixed point solution gives the optimal solution of Eq. (5.2). By taking the derivative of $J(\mathbf{x})$ in Eq. (5.2) w.r.t. \mathbf{x} and setting it to zero, we have

$$(\mathbf{I} - \alpha \widetilde{\mathbf{W}}) \mathbf{x} - (1 - \alpha) \mathbf{b} = \mathbf{0} \Rightarrow \mathbf{x} = \alpha \widetilde{\mathbf{W}} \mathbf{x} + (1 - \alpha) \mathbf{b} \quad (5.3)$$

By grouping the Kronecker products of K normalized adjacency matrices in $\widetilde{\mathbf{W}}$ into two parts $\mathbf{A}_f = \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_k$, $\mathbf{A}_s = \mathbf{A}_{k+1} \otimes \cdots \otimes \mathbf{A}_K$, $k \in (1, K)$, the fixed point iteration can be seen as the procedure on two input networks with adjacency matrices \mathbf{A}_f and \mathbf{A}_s respectively. We transform the Kronecker products in $\widetilde{\mathbf{W}}$ to matrix products $(\mathbf{A}_f \otimes \mathbf{A}_s) \mathbf{v} = \mathbf{A}_s \mathbf{V} \mathbf{A}_f^\top$, where $\mathbf{v} = \text{vec}(\mathbf{V}) = \mathbf{N} \mathbf{D}^{-1/2} \mathbf{x}$. Thus the fixed point method (referred to as Basic Algorithm) can be applied to obtain the solution \mathbf{x} . In a nutshell, we can show that the Basic Algorithm (summarized in Algorithm 5.1) converges to the closed-form solution of Eq. (5.3) with a polynomial time complexity w.r.t. the number of nodes and edges of the input networks.

Note that Basic Algorithm is not the major focus of this paper because of its high complexity. The Basic Algorithm is summarized in Algorithm 5.1.

Algorithm 5.1 Basic Algorithm

Input: K Adjacency matrices of input networks $\mathbf{A}_1, \dots, \mathbf{A}_K$, anchor link vector $\mathbf{b} = \text{vec}(\mathbf{B})$, node or edge attribute matrix \mathbf{N} , \mathbf{E} if available, maximum iteration number t_{max} ;

Output: The solution vector \mathbf{x} of Eq. (5.3).

- 1: Initialize $\mathbf{A}_f, \mathbf{A}_s$ by $k \in [1, K - 1]$, \mathbf{x} , and $t = 1$;
 - 2: Compute $\mathbf{E}_f^q = \bigotimes_{i=1}^k \mathbf{E}_i^q$, $\mathbf{E}_s^q = \bigotimes_{i=k+1}^K \mathbf{E}_i^q, \forall q \in [1, Q]$;
 - 3: **while** $t < t_{max}$ **do**
 - 4: Compute $\mathbf{V} = \text{reshape}(\mathbf{N}\mathbf{D}^{-1/2}\mathbf{x}, n^{K-k}, n^k)$;
 - 5: $\mathbf{x} \leftarrow \alpha\mathbf{D}^{-1/2}\mathbf{N}\text{vec}(\sum_{q=1}^Q (\mathbf{E}_s^q \odot \mathbf{A}_s)\mathbf{V}(\mathbf{E}_f^q \odot \mathbf{A}_f)^\top) + (1 - \alpha)\mathbf{b}$;
 - 6: Set $t \leftarrow t + 1$;
 - 7: **end while**
 - 8: Return \mathbf{x}
-

Corollary 5.1. Convergence and Optimality of the Basic Algorithm. The Algorithm 5.1 converges to the closed form solution $(1 - \alpha)(\mathbf{I} - \alpha\tilde{\mathbf{W}})^{-1}\mathbf{b}$, which is the global minimum of Eq. (5.3).

Proof. See Theorem 1 in [30]. Omitted for brevity.

QED.

We prove the convexity of the objective function.

Lemma 5.1. The objective function in Eq. (5.2) is convex.

Proof. Since the gradient of Eq. (5.2) is $\nabla J(\mathbf{x}) = 2\alpha(\mathbf{I} - \tilde{\mathbf{W}})\mathbf{x} + 2(1 - \alpha)(\mathbf{x} - \mathbf{b}) = 2(\mathbf{I} - \alpha\tilde{\mathbf{W}})\mathbf{x} + 2(1 - \alpha)\mathbf{b}$. The Hessian matrix of objective function in Eq. (5.2) is $\mathbf{H}(J(\mathbf{x})) = 2(\mathbf{I} - \alpha\tilde{\mathbf{W}})$. Since this matrix is strictly diagonal dominant when $\alpha \in (0, 1)$ and also positive definite. Eq. (5.2) is hence convex.

QED.

Sylvester Tensor Equation. In order to speed-up and scale-up the computation, we reformulate Eq. (5.3) as follows. Note that Eq. (5.3) is a linear system w.r.t. vector variable \mathbf{x} . Considering the definition of \mathbf{W} , the linear system in Eq. (5.3) can be re-written as the following *Sylvester tensor equation* by the property of tensor mode product and Kronecker product (i.e., $\mathcal{X} \times_1 \tilde{\mathbf{A}}_K \times_2 \cdots \times_K \tilde{\mathbf{A}}_1 \Leftrightarrow (\tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K)\text{vec}(\mathcal{X})$ [164]).

$$\mathcal{X} - \alpha \sum_{o,p,q} \mathcal{X} \times_1 \tilde{\mathbf{A}}_K^{(o,p,q)} \times_2 \cdots \times_K \tilde{\mathbf{A}}_1^{(o,p,q)} - (1 - \alpha)\mathbf{B} = \mathbf{0} \quad (5.4)$$

where $\tilde{\mathbf{A}}_i^{(o,p,q)} = (\mathbf{D}_i^{-1/2} \mathbf{N}_i^p)(\mathbf{E}_i^o \odot \mathbf{A}_i)(\mathbf{D}_i^{-1/2} \mathbf{N}_i^q)$. When neither node nor edge attributes are available (i.e., plain networks), Eq. (5.4) degenerates into the following Sylvester tensor equation.

$$\mathcal{X} - \alpha \mathcal{X} \times_1 \tilde{\mathbf{A}}_K \times_2 \cdots \times_K \tilde{\mathbf{A}}_1 - (1 - \alpha) \mathcal{B} = \mathbf{0} \quad (5.5)$$

where $\tilde{\mathbf{A}}_i = (\mathbf{D}_i^{-1/2}) \mathbf{A}_i (\mathbf{D}_i^{-1/2})$.

Next, we will present two fast algorithms to solve Eq. (5.5) and Eq. (5.4), respectively.

5.2.3 SYTE-Fast-P for Plain Networks

In this section, we present the proposed fast algorithm, SYTE-Fast-P for solving Eq. (5.5) (i.e., plain networks).

Intuitions and Key Ideas. In order to solve Eq. (5.5) efficiently, the key idea is to decompose its corresponding linear system into a series of subsystems, and then to solve each of them by a tensorized Krylov subspace method. To simplify the description, we absorb the scalar α and $1 - \alpha$ of Eq. (5.3) into the matrix $\widetilde{\mathbf{W}}$ and vector \mathbf{b} respectively to have the following concise equation.

$$(\mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K) \mathbf{x} = \mathbf{b} \quad (5.6)$$

where we let $\mathbf{b} := -(1 - \alpha) \mathbf{b}$, and $\tilde{\mathbf{A}}_i := \alpha^{1/K} \tilde{\mathbf{A}}_i, \forall i \in [1, K]$. The coefficient matrix of this linear system is strictly diagonally dominant and positive definite. Therefore, the system is solvable with a unique solution [115]. However, the coefficient matrix in this linear system contains the Kronecker product. A direct solver such as Krylov subspace method would cost at least $O(N^2)$ where $N = n^K \gg n$ is the dimension of the linear system in Eq. (5.6), due to the construction of the orthonormal basis of the Krylov subspace [25]. To address this issue, the key idea is to use a tensorized Krylov subspace, and let the solution tensor reside in the tensorized Krylov subspace, so as to avoid explicit calculation of the orthonormal basis. In this way, the solution can be represented in a Tucker decomposition form without the explicit calculation, which will reduce the time complexity from $O(N^2)$ to $O(c_1 m + c_2 l^K)$ and reduce the space complexity from $O(N + M)$ to $O(c_3 n + K m + l^{2K})$. Here, c_1, c_2, c_3, l are small constants, and K can also be regarded as a constant because it is often much smaller than both n and m (see details below).

SyTE-Fast-P Algorithm. Firstly, notice that tensor \mathcal{B} contains s non-zero entries (i.e., s known anchor links). We represent \mathcal{B} as the summation of the outer product of the indicator vectors $\mathcal{B} = \sum_{i=1}^s \mathbf{b}_K^{(i)} \circ \cdots \circ \mathbf{b}_1^{(i)}$ where $\mathbf{b}_j^{(i)}$ is an indicator vector of the i -th non-zero entry in \mathcal{B} corresponding to mode j . Tensor \mathcal{B} can be further re-written as its vector form $\mathbf{b} =$

$\sum_{i=1}^s \mathbf{b}_1^{(i)} \otimes \cdots \otimes \mathbf{b}_K^{(i)}$. For the example in Figure 5.1, the anchor link tensor \mathbf{B} only contains one non-zero entry at $(4, 4, 4)$. It can be shown that $\mathbf{B} = [0, 0, 0, 1]^\top \circ [0, 0, 0, 1]^\top \circ [0, 0, 0, 1]^\top$, and $\mathbf{b} = [0, 0, 0, 1]^\top \otimes [0, 0, 0, 1]^\top \otimes [0, 0, 0, 1]^\top$. Then, Eq. (5.6) is decomposed into the following subsystems.

$$(\mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K) \mathbf{x}_i = \bigotimes_{j=1}^K \mathbf{b}_j^{(i)} \quad (i = 1, \dots, s) \quad (5.7)$$

Secondly, instead of directly constructing the Krylov subspace $\mathcal{K}_L(\mathbf{A}_\times, \mathbf{b})$ where \mathbf{A}_\times denotes $\mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K$, and L is the dimension of the Krylov subspace, we construct the Krylov subspaces for $\tilde{\mathbf{A}}_1, \dots, \tilde{\mathbf{A}}_K$ separately as $\mathcal{K}_{l_1}(\tilde{\mathbf{A}}_1, \mathbf{b}_1), \dots, \mathcal{K}_{l_K}(\tilde{\mathbf{A}}_K, \mathbf{b}_K)$. Then, we use the Kronecker product of these K Krylov subspaces as the new subspace. Using the same notation as [165], we define the tensorized Krylov subspace as follows.

$$\mathcal{K}_{\mathfrak{L}}^{\otimes}(\mathbf{A}_\times, \mathbf{b}) := \text{span}(\mathcal{K}_{l_1}(\tilde{\mathbf{A}}_1, \mathbf{b}_1) \otimes \cdots \otimes \mathcal{K}_{l_K}(\tilde{\mathbf{A}}_K, \mathbf{b}_K)) \quad (5.8)$$

where $\mathfrak{L} = (l_1, \dots, l_K)$ is a multi-index. For each Krylov subspace $\mathcal{K}_{l_i}(\tilde{\mathbf{A}}_i, \mathbf{b}_i)$, we use the standard Arnoldi method³ to obtain the orthonormal basis represented in \mathbf{U} with orthonormal columns. Specifically, the Arnoldi method gives the Hessenberg matrix $\tilde{\mathbf{H}}_i = \mathbf{U}_{l_i+1}^\top \tilde{\mathbf{A}}_i \mathbf{U}_{l_i}$ of size $(l_i + 1) \times l_i$. Note that l_i is often much smaller than n_i (i.e., $l_i \ll n_i$). We can prove that (1) $\bigotimes_{i=1}^K \mathbf{U}_{l_i}$ forms the orthonormal basis of $\mathcal{K}_{\mathfrak{L}}^{\otimes}(\mathbf{A}_\times, \mathbf{b})$, and (2) the original Krylov subspace $\mathcal{K}_L(\mathbf{A}_\times, \mathbf{b})$ is contained in the tensorized Krylov subspace $\mathcal{K}_{\mathfrak{L}}^{\otimes}(\mathbf{A}_\times, \mathbf{b})$ [165].

Thirdly, we can further prove that by using a tensorized Krylov subspace based generalized minimal residual method, each subsystem $(\mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K) \mathbf{x}_i = \bigotimes_{j=1}^K \mathbf{b}_j^{(i)}$ in Eq. (5.7) can be solved by the following linear system, whose scale is much smaller than the original linear system:

$$\left(\bigotimes_{i=1}^K \mathbf{I}_{l_i+1, l_i} - \bigotimes_{i=1}^K \tilde{\mathbf{H}}_i \right) \mathbf{y} = \bigotimes_{i=1}^K \mathbf{U}_{l_i+1}^\top \mathbf{r}_0 \quad (5.9)$$

Note that the coefficient matrix has a Hessenberg-like structure. Thus, it can be solved by the back-substitute method. Putting everything together, the proposed SyTE-Fast-P algorithm is presented in Algorithm 5.2. We use the same dimension for all the Krylov subspaces for notation simplicity. Note that $\mathbf{x} = \mathbf{x}_1 + \cdots + \mathbf{x}_K$, where each $\mathbf{x}_i = \bigotimes_{j=1}^K \mathbf{U}_{l_j}^{(i)} \mathbf{y}_i$, and $\mathcal{X} = \text{reshape}(\mathbf{x}, n_K, \dots, n_1)$.

SyTE-Fast-P* Algorithm. Here, we present another fast algorithm for solving the Sylvester tensor equation of plain network (Eq. (5.5)). First, since each $\tilde{\mathbf{A}}_i$ in Eq. (5.6)

³Lanczos algorithm could also be adopted here alternatively.

Algorithm 5.2 SYTE-Fast-P Algorithm

Input: K normalized adjacency matrices of input networks $\tilde{\mathbf{A}}_1, \dots, \tilde{\mathbf{A}}_K$, tensor \mathcal{B} or $\mathbf{b} = \text{vec}(\mathcal{B})$ with s known multi-way anchor links, Krylov subspace size $l > 0$;

Output: The solution tensor \mathcal{X} of Eq. (5.6).

- 1: Decompose \mathbf{b} for Eq. (5.7) to obtain $\{\mathbf{b}_j^{(i)}\}_{j \in [1, K], i \in [1, s]}$;
 - 2: **for** $i = 1, \dots, s$ **do**
 - 3: Initialize \mathbf{x}_i as a zero vector;
 - 4: **for** $j = 1, \dots, K$ **do**
 - 5: Construct $\mathcal{K}_l(\mathbf{I}_j - \tilde{\mathbf{A}}_j, \mathbf{b}_j)$;
 - 6: Obtain $\tilde{\mathbf{H}}_j^{(i)}$, $\mathbf{U}_{l_j}^{(i)}$, and $\mathbf{U}_{l_j+1}^{(i)}$;
 - 7: **end for**
 - 8: Solve Eq. (5.9) to obtain \mathbf{y}_i ;
 - 9: **end for**
 - 10: Return implicit solution $\{\mathbf{y}_i, \{\mathbf{U}_{l_j}^{(i)}\}_{j=1}^K\}_{i=1}^s$.
-

is diagonalizable (i.e., real symmetric matrix), we take the eigen-decomposition of each $\tilde{\mathbf{A}}_i = \mathbf{Q}_i \mathbf{\Lambda}_i \mathbf{Q}_i^T$, in which \mathbf{Q}_i is a matrix with each column as an eigenvector. In this case, it can be easily proved that the Eq. (5.6) can be written as:

$$(\mathbf{I} - \mathbf{\Lambda}_1 \otimes \dots \otimes \mathbf{\Lambda}_K) \mathbf{y} = \mathbf{c} \quad (5.10)$$

where $\mathbf{c} = \mathbf{Q}^T \mathbf{b}$ and $\mathbf{x} = \mathbf{Q} \mathbf{y}$, and $\mathbf{Q} = \mathbf{Q}_1 \otimes \dots \otimes \mathbf{Q}_K$. Note that Eq. (5.10) is very easy to solve because the coefficient matrix is diagonal. If we use full eigen-decomposition for each $\tilde{\mathbf{A}}_i$, there would be no approximation error. However, the time complexity of calculating \mathbf{x} from intermediate variable \mathbf{y} would be $O(N^2)$. Since the adjacency matrices are usually sparse and low-rank, we could use rank- r ($r \ll n$) approximation on the eigen-decomposition of each $\tilde{\mathbf{A}}_i$. Then the linear system in Eq. (5.10) becomes much smaller ($r^K \times r^K$ instead of $n^K \times n^K$). For notation simplicity, we use the same r for each $\tilde{\mathbf{A}}_i$. The time complexity of calculating \mathbf{y} is $O(r^K)$. Adding the time complexity of eigen-decomposition and calculating \mathbf{c} , the overall time complexity is reduced to $O(Krn^2 + r^K + r^K n^K)$. The proposed SYTE-Fast-P* algorithm is summarized in Algorithm 5.3.

For simplicity, we assume the ranks of eigen-decomposition are the same for each $\tilde{\mathbf{A}}_i$ in line 2. The intermediate solution \mathbf{y} is solved in line 4, and the implicit representation of solution is returned and stored in line 5, which will significantly reduce the space complexity. \mathbf{x} is calculated as $\mathbf{x} = \bigotimes_{j=1}^K \mathbf{Q}_j \mathbf{y}$.

With the proposal of Algorithm 5.3 on plain networks, the SYTE-Fast-A, which uses SYTE-Fast-P in line 4 of Algorithm 5.4, has another variant that instead uses SYTE-Fast-P*. We name it SYTE-Fast-A*.

Algorithm 5.3 SYTE-Fast-P* Algorithm

Input: K Normalized adjacency matrices of input networks $\tilde{\mathbf{A}}_1, \dots, \tilde{\mathbf{A}}_K$, multi-way anchor link tensor \mathbf{B} , approximation rank r ;

Output: The solution tensor \mathcal{X} of Eq. (5.6).

- 1: **for** $i = 1, \dots, K$ **do**;
 - 2: Conduct top- r eigen-decomposition on $\tilde{\mathbf{A}}_i$ for $\mathbf{Q}_i, \mathbf{\Lambda}_i$;
 - 3: **end for**
 - 4: Calculate $\mathbf{c} = \bigotimes_{j=1}^K \mathbf{Q}_j^\top \mathbf{b}$, and solve Eq. (5.10) to obtain \mathbf{y} ;
 - 5: Return implicit solution $\{\mathbf{y}, \mathbf{Q}_j\}_{j=1}^K$.
-

Proofs and Analysis. We give the following theorem for the complexity of the proposed SYTE-Fast-P algorithm. In the analysis of complexity, for notation simplicity, we assume all input networks share the same number of nodes n and number of edges m .

Theorem 5.1. Complexity of SyTE-Fast-P. The time complexity of Algorithm 5.2 is $O(sKlm + sl^K)$. The space complexity of Algorithm 5.2 is $O(Km + l^{2K} + Kln)$.

Proof. The Arnoldi process takes $O(lm)$ for one iteration, and the number of total iteration is sK . Thus the complexity for Arnoldi process is $O(sKlm)$. Solving s linear systems of Eq. (5.9) takes $O(sl^K)$, which is linear w.r.t. the size of the linear system, thanks to the back-substitute method. The overall time complexity for SYTE-Fast-P is $O(sKlm + sl^K)$.

For space complexity, storing K adjacency matrices takes $O(Km)$. Storing the Hessenberg matrix for each Eq. (5.9) in the outer iteration takes $O(l^{2K})$. Storing the orthonormal basis $\mathbf{U}_{l_j}^{(i)}$ for the outer iteration takes $O(Kln)$. Overall, the space complexity is $O(Km + l^{2K} + Kln)$ for SYTE-Fast-P. QED.

Remark. Since K, l, s are usually much smaller than m, n (K, l, s are treated as *constants*⁴ in the big-O notation), Algorithm 5.2 has a much smaller time complexity than the Basic Algorithm (Algorithm 5.1) whose time complexity is $O(Qm^{\lfloor K/2 \rfloor} n^{\lfloor K/2 \rfloor} \cdot t_{max} + n^K)$. Furthermore, both the space and time complexities of Algorithm 5.2 is *linear* w.r.t. the size (i.e., the number of nodes and edges) of input networks.

We then give the following theorem for the complexity of the proposed SYTE-Fast-P* algorithm. In the analysis of complexity, for notation simplicity, we assume all input networks share the same number of nodes n and number of edges m .

Theorem 5.2. Complexity of SyTE-Fast-P*. The time complexity of Algorithm 5.3 is $O(Krn^2 + r^K + r^K n^K)$. The space complexity of Algorithm 5.3 is $O(Km + Knr)$.

⁴We assume that the number of input networks K is a small non-variant constant.

Proof. SyTE-Fast-P:* K top- r eigen-decomposition takes $O(Krn^2)$. Solving Eq. (5.10) takes linear time complexity w.r.t. the linear system size, $O(r^K)$. Calculating \mathbf{c} takes $O(r^K n^K)$. Overall, the time complexity for SyTE-Fast-P* is $O(Krn^2 + r^K + r^K n^K)$.

For space complexity, storing K adjacency matrices takes $O(Km)$, and storing K \mathbf{Q} matrices of eigenvectors takes $O(Knr)$. Overall, the space complexity for SyTE-Fast-P* is $O(Km + Knr)$. QED.

Since K, r are usually much smaller than m, n (K, r are treated as *constants* in the big-O notation), both Algorithm 5.2 and Algorithm 5.3 have a much smaller time complexity than the Basic Algorithm (Algorithm 5.1) whose time complexity is $O(Qm^{\lfloor K/2 \rfloor} n^{\lfloor K/2 \rfloor} \cdot t_{max} + n^K)$.

5.3 MULTI-WAY ASSOCIATION ON ATTRIBUTED NETWORKS

In this section, following the problem definition and convex formulation of Section 5.2, we present a fast algorithm to solve the Sylvester equation on attributed networks (i.e. Eq. (5.4)).

5.3.1 Intuitions and Key Ideas

Here, we present a fast solver when the node attributes are available. Notice that SyTE-Fast-P can not be directly applied because the \mathbf{W} matrix in Eq. (5.2) contains summations of Kronecker products. To address this issue, we have the following key observations.

Firstly, WLOG, assume that nodes in each network are reordered such that the nodes with the same node attributes have adjacent indices. For the example in Figure 5.1, assume that nodes $\{1, 2\}$ and $\{3, 4\}$ in $G_i, \forall i \in [1, 3]$ have the same attributes respectively. We observe that the solution tensor of Eq. (5.4) has a block-diagonal structure, which means that the non-zero entries in the solution tensor \mathcal{X} only exist in the diagonal block tensors. Intuitively, this is because the diagonal blocks in the solution tensor correspond to nodes across networks with the same node attributes, meanwhile the off-diagonal blocks correspond to nodes across networks with different node attributes. This indicates that Eq. (5.4) could be decomposed into a series of subsystems by node attributes.

Secondly, based on the above observation, we only need to solve the diagonal tensors by block coordinate descent (*BCD*) method. The off-diagonal entries in \mathcal{X} can be set equal to the corresponding entries with the same indices in \mathcal{B} . The linear system for the example in

Figure 5.1 can be decomposed into:

$$\begin{aligned} \boldsymbol{\mathcal{X}}^{1,1,1} - [\boldsymbol{\mathcal{X}}^{1,1,1} \times_1 \tilde{\mathbf{A}}_1^{11} \dots \times_3 \tilde{\mathbf{A}}_3^{11} + \underbrace{\boldsymbol{\mathcal{X}}^{2,2,2} \times_1 \tilde{\mathbf{A}}_1^{12} \dots \times_3 \tilde{\mathbf{A}}_3^{12}}_{\mathcal{C}_{2,2,2}^{1,1,1}}] &= \mathbf{B}^{1,1,1} \\ \boldsymbol{\mathcal{X}}^{2,2,2} - [\boldsymbol{\mathcal{X}}^{2,2,2} \times_1 \tilde{\mathbf{A}}_1^{22} \dots \times_3 \tilde{\mathbf{A}}_3^{22} + \underbrace{\boldsymbol{\mathcal{X}}^{1,1,1} \times_1 \tilde{\mathbf{A}}_1^{21} \dots \times_3 \tilde{\mathbf{A}}_3^{21}}_{\mathcal{C}_{1,1,1}^{2,2,2}}] &= \mathbf{B}^{2,2,2} \end{aligned}$$

with $\boldsymbol{\mathcal{X}}^{ijk} = \mathbf{B}^{ijk}, \forall i \neq j$ or $j \neq k$ or $i \neq k$. $\boldsymbol{\mathcal{X}}^{1,1,1}$ and $\boldsymbol{\mathcal{X}}^{2,2,2}$ denote the $(1, 1, 1)$ -th and $(2, 2, 2)$ -th tensor block respectively (similar notation for \mathbf{B}). $\tilde{\mathbf{A}}^{11} = \mathbf{D}^{-1/2} \mathbf{N}^1 \mathbf{A} \mathbf{N}^1 \mathbf{D}^{-1/2}$ denotes the normalized adjacency matrix, filtered by the first node attribute. The parameter α , together with $(1-\alpha)$ on \mathbf{B} , is absorbed into tensors for notation simplicity. $\mathcal{C}_{1,1,1}^{2,2,2}$ represents the contribution of block variable $\boldsymbol{\mathcal{X}}^{1,1,1}$ to $\boldsymbol{\mathcal{X}}^{2,2,2}$, and the similar notation applies for $\mathcal{C}_{2,2,2}^{1,1,1}$.

Thirdly, *BCD* requires that each $\boldsymbol{\mathcal{X}}^{i\dots i}$ be computed explicitly (e.g., by the Basic Algorithm), because each block variable is needed to calculate other block variables. Although *BCD* is faster than applying the Basic Algorithm on the whole variable tensor, the computational complexity is still polynomial. To address this issue, the idea is to omit the contribution of diagonal blocks from other subsystems (e.g. $\mathcal{C}_{1,1,1}^{2,2,2}$ and $\mathcal{C}_{2,2,2}^{1,1,1}$) for each subsystem. In this way, we only need one single iteration by Algorithm 5.2 to approximately solve each diagonal block independently.

5.3.2 SYTE-Fast-A Algorithm

Generally speaking, Eq. (5.4) with node attributes can be decomposed as: $\boldsymbol{\mathcal{X}}^{i\dots i} - \sum_{j=1} \boldsymbol{\mathcal{X}}^{j\dots j} \times_1 \tilde{\mathbf{A}}_1^{ij} \dots \times_K \tilde{\mathbf{A}}_K^{ij} = \mathbf{B}^{i\dots i}$, where the off-diagonal variant blocks are equal to the corresponding blocks in \mathbf{B} . It can be solved by approximated block coordinate descent as follows. By the exact *BCD*, each iteration should solve:

$$\boldsymbol{\mathcal{X}}^{i\dots i} - \boldsymbol{\mathcal{X}}^{i\dots i} \times_1 \tilde{\mathbf{A}}_1^{ii} \dots \times_K \tilde{\mathbf{A}}_K^{ii} = \hat{\mathbf{B}}^{i\dots i} \quad (5.12)$$

where $\hat{\mathbf{B}}^{i\dots i} = \mathbf{B}^{i\dots i} + \sum_{j \neq i} \boldsymbol{\mathcal{X}}^{j\dots j} \times_1 \tilde{\mathbf{A}}_1^{ij} \dots \times_K \tilde{\mathbf{A}}_K^{ij}$, and it can be viewed as the updated $\mathbf{B}^{i\dots i}$ tensor. If we approximate $\hat{\mathbf{B}}^{i\dots i} = \mathbf{B}^{i\dots i}$, each subsystem can be solved in one single iteration. The SYTE-Fast-A algorithm is summarized in Algorithm 5.4. Similar to Algorithm 5.2, line 6 returns the *implicit* solution of diagonal block variables.

5.3.3 Proofs and Analysis

Next, we provide the complexity analysis of the proposed algorithm. Let m_i , n_i and s_i ($i \in [1, P]$) be the number of edges/nodes in $\tilde{\mathbf{A}}_k^{ii}$ ($k \in [1, K]$), and the number of non-zero

entries in $\mathbf{B}^{i\dots i}$, respectively. For notation simplicity, we assume $m_i, \forall i \in [1, P]$ is the same for each input network. Let l be the subspace size when using Algorithm 5.2 in solving Eq. (5.12).

Theorem 5.3. Complexity of SyTE-Fast-A. The time complexity of Algorithm 5.4 is $O(Km + n + \sum_{i=1}^P (s_i K l m_i + s_i l^K))$. The space complexity of Algorithm 5.4 is $O(PK m_i + K l n_i + l^{2K})$.

Proof. Constructing $\tilde{\mathbf{A}}_1^{ii}, \dots, \tilde{\mathbf{A}}_K^{ii}, \forall 1 \leq j \leq P$ takes $O(Km)$. Calculating $\mathbf{B}^{i\dots i}$ takes $O(n)$ because the tensor \mathbf{B} is rank- n . Solving P equations of Eq. (5.12) by using SyTE-Fast-P takes $O(\sum_{i=1}^P (s_i K l m_i + s_i l^K))$. Overall, the time complexity of SyTE-Fast-A is $O(Km + n + \sum_{i=1}^P (s_i K l m_i + s_i l^K))$.

For space complexity, storing $\tilde{\mathbf{A}}_1^{ii}, \dots, \tilde{\mathbf{A}}_K^{ii}$ takes $O(PK m_i)$. Since in the iteration only one Eq. (5.12) is solved each time, storing Hessenberg matrices and orthonormal basis takes $O(K l n_i + l^{2K})$. The overall space complexity for SyTE-Fast-A is $O(PK m_i + K l n_i + l^{2K})$.

QED.

Algorithm 5.4 SyTE-Fast-A Algorithm

Input: Normalized adjacency matrices $\tilde{\mathbf{A}}_1, \dots, \tilde{\mathbf{A}}_K$; node attribute matrices $\mathbf{N}_1, \dots, \mathbf{N}_K$; Krylov subspace size $l > 0$; tensor \mathbf{B} or $\mathbf{b} = \text{vec}(\mathbf{B})$;

Output: The solution tensor \mathcal{X} of Equation (5.4).

- 1: Construct block matrices $\tilde{\mathbf{A}}_1^{ij}, \dots, \tilde{\mathbf{A}}_K^{ij}$, block tensor $\mathbf{B}^{i\dots i}, \forall 1 \leq i, j \leq P$ by the node attribute matrices $\mathbf{N}_1, \dots, \mathbf{N}_K$;
 - 2: Initialize $\mathcal{X}^{i\dots i}, \forall i \in [1, K]$;
 - 3: **for** $p = 1, \dots, P$ **do**
 - 4: Solve Eq. (5.12) by Algorithm 5.2 to obtain $\{\mathbf{y}_i, \{\mathbf{U}_{l_j}^i\}_{j=1}^K\}_{i=1}^s$;
 - 5: **end for**
 - 6: Return implicit solution $\{\{\mathbf{y}_i^p, \{\mathbf{U}_{l_j}^{i,p}\}_{j=1}^K\}_{i=1}^s\}_{p=1}^P$.
-

Remark 5.1. From Theorem 5.3, note that the number of node attributes has great impact on time and space complexity, since $\sum_{i=1}^P m_i = m$. A larger number of node attribute will lead to smaller complexity for computing each block tensor variable. Also note that the time complexity is much less than the basic method, which is $O(N)$.

We then prove that the subsystems decomposed in Eq. (5.7) can be solved by a much smaller linear system (Eq. (5.9)) as follows.

Theorem 5.4. Subsystem $(\mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \dots \otimes \tilde{\mathbf{A}}_K) \mathbf{x}_i = \bigotimes_{j=1}^K \mathbf{b}_j^{(i)}$ can be solved by first solving a small-scaled linear system $(\bigotimes_{i=1}^K \mathbf{I}_{l_i+1, l_i} - \bigotimes_{i=1}^K \tilde{\mathbf{H}}_i) \mathbf{y} = \bigotimes_{i=1}^K \mathbf{U}_{l_i+1}^\top \mathbf{r}_0$.

Proof. For subsystem $(\mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K)\mathbf{x}_i = \bigotimes_{j=1}^K \mathbf{b}_j^{(i)}$, the initial residual vector $\mathbf{r}_0 = \mathbf{b} - (\mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K)\mathbf{x}_0$, given an initial solution vector \mathbf{x}_0 (which is typically initialized as zero vector). Let the updated solution vector be $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{z}_0$, in which we let $\mathbf{z}_0 \in \mathcal{K}_{\mathcal{G}}^{\otimes}$. So $\mathbf{z}_0 = \bigotimes_{i=1}^K \mathbf{U}_{l_i}\mathbf{y}$. The updated residual to be minimized is as follows.

$$\mathbf{r}_1 = \mathbf{r}_0 - (\mathbf{I} - \bigotimes_{i=1}^K \mathbf{A}_i)(\bigotimes_{i=1}^K \mathbf{U}_{l_i})\mathbf{y} = \mathbf{r}_0 - (\bigotimes_{i=1}^K \mathbf{U}_{l_i})\mathbf{y} + (\bigotimes_{i=1}^K \mathbf{U}_{l_{i+1}})(\bigotimes_{i=1}^K \tilde{\mathbf{H}}_i)\mathbf{y} \quad (5.13)$$

Recall that the second equation above is due to the Arnoldi process, which gives $\tilde{\mathbf{H}}_i = \mathbf{U}_{l_{i+1}}^T \mathbf{A}_i \mathbf{U}_{l_i}$. Minimizing the norm of the updated residual gives us:

$$\begin{aligned} \min_{\mathbf{y}} \|\mathbf{r}_1\|_2^2 &= \min_{\mathbf{y}} \left\| \bigotimes_{i=1}^K \mathbf{U}_{l_{i+1}} (\bigotimes_{i=1}^K \mathbf{U}_{l_{i+1}}^T \mathbf{r}_0 - \bigotimes_{i=1}^K \mathbf{I}_{l_{i+1}, l_i} \mathbf{y} + \bigotimes_{i=1}^K \tilde{\mathbf{H}}_i \mathbf{y}) \right\|_2^2 \\ &= \min_{\mathbf{y}} \left\| \bigotimes_{i=1}^K \mathbf{U}_{l_{i+1}}^T \mathbf{r}_0 - \bigotimes_{i=1}^K \mathbf{I}_{l_{i+1}, l_i} \mathbf{y} + \bigotimes_{i=1}^K \tilde{\mathbf{H}}_i \mathbf{y} \right\|_2^2 \end{aligned} \quad (5.14)$$

where the second step is because $\bigotimes_{i=1}^K \mathbf{U}_{l_{i+1}}$ is a matrix with all columns being orthogonal with each other. In the above equation $\mathbf{I}_{l_{i+1}, l_i} = [\delta_{i,j}]_{1 \leq i \leq l_{i+1}, 1 \leq j \leq l_i}$, in which $\delta_{i,j}$ is the Kronecker δ -function, which is an identity like matrix with 1 in "diagonal" entries. Solving the minimization problem in Eq. (5.14) is actually equal to solving a smaller scaled linear system, compared to the original large linear system:

$$\left(\bigotimes_{i=1}^K \mathbf{I}_{l_{i+1}, l_i} - \bigotimes_{i=1}^K \tilde{\mathbf{H}}_i \right) \mathbf{y} = \bigotimes_{i=1}^K \mathbf{U}_{l_{i+1}}^T \mathbf{r}_0 \quad (5.15)$$

Note that $\mathbf{x} = \mathbf{x}_1 + \cdots + \mathbf{x}_K$, where each $\mathbf{x}_i = \bigotimes_{j=1}^K \mathbf{U}_{l_j}^{(i)} \mathbf{y}_i$, and the solution tensor $\mathcal{X} = \text{reshape}(\mathbf{x}, n_K, \dots, n_1)$, which does not need to be explicitly calculated in our algorithm. QED.

5.3.4 Sensitivity Analysis

In this section, we analyze the sensitivity of the linear system formulated in Eq. (5.3). To be specific, we aim to understand how the solution of Eq. (5.3) will be impacted if some edges of the input networks are changed, due to either random noise or adversarial attacks (e.g., edge removal [166]). Given K networks G_1, \dots, G_K and the budget for edge perturbation in each network p_1, \dots, p_K . Let $\mathbf{A}_{\times} = \mathbf{I} - \tilde{\mathbf{A}}_1 \otimes \cdots \otimes \tilde{\mathbf{A}}_K$, $\hat{\mathbf{A}} = \mathbf{A}_1 \otimes \cdots \otimes \mathbf{A}_K$,

we have:

$$(\mathbf{A}_\times + \Delta\mathbf{A}_\times)(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} \quad (5.16)$$

where $\Delta\mathbf{A}_\times$ and $\Delta\mathbf{x}$ are the perturbations to \mathbf{A}_\times and solution \mathbf{x} respectively. We present the following theorem:

Theorem 5.5. The relative change after edge perturbation satisfies:

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\eta}{1 - \epsilon} \left(\prod_{i=1}^K m_i - \prod_{i=1}^K (m_i - 2p_i) \right)^{1/2} \quad (5.17)$$

where $\|\mathbf{A}_\times\|_F = \epsilon < 1$, $\eta = \|\mathbf{D}^{-1/2}\|_F^2$. m_i is the number of edges in network G_i .

Proof. Based on Eq. (5.16), we have $\Delta\mathbf{x} \approx -\mathbf{A}_\times^{-1}\Delta\mathbf{A}_\times\mathbf{x}$ by dropping the high-order small term $\Delta\mathbf{A}_\times\Delta\mathbf{x}$. The relative change after edge removal is as follows:

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} = \frac{\|\mathbf{A}_\times^{-1}\Delta\mathbf{A}_\times\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\|\mathbf{A}_\times^{-1}\| \|\Delta\mathbf{A}_\times\| \|\mathbf{x}\|}{\|\mathbf{x}\|} = \kappa(\mathbf{A}_\times) \frac{\|\Delta\mathbf{A}_\times\|}{\|\mathbf{A}_\times\|} \quad (5.18)$$

where $\kappa(\mathbf{A}_\times) = \|\mathbf{A}_\times\| \|\mathbf{A}_\times^{-1}\|$ is the condition number of matrix \mathbf{A}_\times . Note that \mathbf{A}_\times is invertible since it is nonsingular. Since $\|\mathbf{A}_\times\|_\infty < 1$, we have $\mathbf{A}_\times^{-1} = \sum_{j=0}^{\infty} (\mathbf{A}_\times)^j$. Therefore,

$$\|\mathbf{A}_\times^{-1}\|_F \leq \sum_{j=0}^{\infty} \epsilon^j = \frac{1}{1 - \epsilon} \quad (5.19)$$

$\|\Delta\mathbf{A}_\times\| = \|\mathbf{A}_\times - \mathbf{A}'_\times\| \leq \|\mathbf{D}^{-1/2}(\hat{\mathbf{A}} - \hat{\mathbf{A}}')\mathbf{D}^{-1/2}\|$, where \mathbf{A}'_\times and $\hat{\mathbf{A}}'$ are perturbed \mathbf{A}_\times and $\hat{\mathbf{A}}$, respectively. Thus we have:

$$\|\Delta\mathbf{A}_\times\| \leq \|\mathbf{D}^{-1/2}\|_F^2 \|\hat{\mathbf{A}} - \hat{\mathbf{A}}'\| = \eta \left(\prod_{i=1}^K m_i - \prod_{i=1}^K (m_i - 2p_i) \right)^{1/2} \quad (5.20)$$

where $\prod_{i=1}^K m_i - \prod_{i=1}^K (m_i - 2p_i)$ is the number of non-zero entries in $\Delta\mathbf{A}_\times$. Plugging Eq. (5.19) (5.20) into Eq. (5.18) leads to Eq. (5.17). QED.

From Theorem 5.5, we can see that the relative change of the solution \mathbf{x} is bounded by the norm of the perturbed matrix $\Delta\mathbf{A}_\times$. One implication for adversarial attacking (e.g., removing certain edges to maximally alter the solution \mathbf{x}) of this bound is as follows. Intuitively, a good attacking strategy might be to remove the edges in each $\tilde{\mathbf{A}}_i$ with high weights, since this will lead to the largest relative norm change (i.e., a larger upper bound in Theorem 5.5).

5.4 EXPERIMENTAL EVALUATIONS

In this section we present the experimental results to answer the following questions:

- **Q1 Effectiveness.** How effective and accurate are the proposed SYTE methods for inferring multi-way association?
- **Q2 Efficiency.** How fast and scalable are the proposed SYTE methods?

5.4.1 Experimental Setup

Datasets. We use five datasets for evaluations whose statistics is summarized in Table 5.2.

We use five datasets for evaluations as follows:

Table 5.2: Datasets Summary

Dataset Name	Category	# of Nodes	# of Edges
<i>DBLP</i>	Co-authorship	1,013	3,244
<i>Arxiv</i>	Academic network	2,908	3,551
<i>Douban</i>	User relationship	3,384	6,556
<i>Aminer</i>	Academic network	1,274,360	4,756,194
Dataset Name	# of Users	# of Artists	# of Tags
<i>LastFm</i>	15,154	2,982	4,144

DBLP is a co-authorship network. Nodes represents authors while links represents co-authorship relation. The original dataset contains 42,252 nodes and 210,320 edges [167].

LastFm is a dataset for recommendation. It contains user-user friendship relation, user-artist listening relation, artist-tag categorization relation and artist profile. The original dataset contains 1,982 users, 17,632 artists and 11,946 tags [168].

Douban includes the users’ friend relation in the online social network and offline activities, which share overlapping users. It contains 50k users and 5M edges in the original data. [129].

*Arxiv*⁵ is a co-authorship network from two physical and one mathematical domains. Nodes represents authors and links represents co-authorship relation.

Aminer is an academic social network. Undirected edges represent co-authorship relationship. The whole dataset contains 1,274,360 nodes and 4,756,194 edges [129].

Comparison methods. In total, we evaluate 12 methods, including the proposed SYTE algorithms. For one-to-one multi-network alignment, we compare with *CLF* [169], *FINAL* [30] and *IsoRank* [163]. For multi-network node retrieval, we compare with *REGAL* [114], *CrossMNA* [18], *FINAL* [30], and *IsoRank* [163]. For high-order recommendation, we compare with *nNTF* (non-negative tensor factorization), *NTF* (Neural Tensor Factorization)

⁵<https://comunelab.fbk.eu/data.php>

[170], and *wiZAN-Dual* (Dual-Regularized One-Class Collaborative Filtering) [171]. For scalability study, we compare the proposed fast methods *SYTE-Fast-P* and *SYTE-Fast-A* with two classic Sylvester equation solvers, including *FP* (Fixed Point method) and *CG* (Conjugate Gradient method) [115].

Evaluation Tasks. We design the following tasks for evaluations.

Task 1. Multi-network alignment. We first conduct multi-network alignment on three networks, which is different from traditional pair-wise network alignment. We use the following datasets. Three networks from *Arxiv* (two physical domains and one mathematical domain), three networks from *DBLP* (the original *DBLP* network and two permuted *DBLP* networks with 5% randomly added edge noises), and three networks from *Douban* (two *Douban* online networks with following and messaging relation respectively and one offline network).

Task 2. Multi-network node retrieval. In order to compare with some multi-network alignment baseline methods which do not support one-to-one alignment, we design a multi-network node retrieval experiment which is often referred to as ‘soft alignment’ [18, 114]. Given three networks and a node from one network, the goal of multi-network node retrieval is to return a ranking tuple list such that the similar nodes from other networks would appear in high ranks of the tuple list.

Task 3. High-order recommendation. To further evaluate the effectiveness of multi-way association, we conduct high-order recommendation. Traditional recommendation only recommend items to users, but we conduct high-order recommendation task to recommend tuples with (artist, tag) to users simultaneously on *LastFm*. The original dataset contains user-user interaction network, the (user, artist, tag) tuples reflecting the user’s listening behavior, the (artist, tag) tuples reflecting the categories of artists, and (user, artist) tuple reflecting the users’ artist preference. The artist-artist network is constructed by the the artists’ cosine similarities calculated from (user, artist) tuples. The tag-tag network is constructed by calculating the cosine similarities of tags pairs in the (artist, tag) tuples. Note that the attributes are not used in this task.

Hardware and software. All of the datasets are public. All experiments are performed on a machine with Intel(R) Core(TM) i7-9800X CPU with 3.80 GHz and 64.0 GB RAM. The algorithms are programmed with MATLAB R2019a with parallel computing.

Adjustment of Baseline methods. For pairwise network alignment methods in multi-network alignment task (*FINAL*, *IsoRank*, *CLF*), the alignment is first conducted on each pair of networks independently, and then the node alignments of each pair of networks are merged together to compare the multi-network alignment performance. The same strategy is used for pairwise methods in multi-network node retrieval task. For high-order recommendation task, *wiZAN-Dual* is conducted on user-user relation network with user-artist

network, and user-user network with user-tag network, respectively. The recommendation result for each user is then merged together for high-order recommendation.

Other implementation details. For multi-network alignment, we implement a high-order greedy match algorithm. The multi-way association tensor can be transform to a high-order 0-1 tensor, in which each fiber contains at most one non-zero entry. For multi-network node retrieval task, given one node i_K from network G_K , the ranking list of the nodes from the rest of networks is calculated by sorting the slice of $\mathcal{X}(i_K, :, \dots :)$.

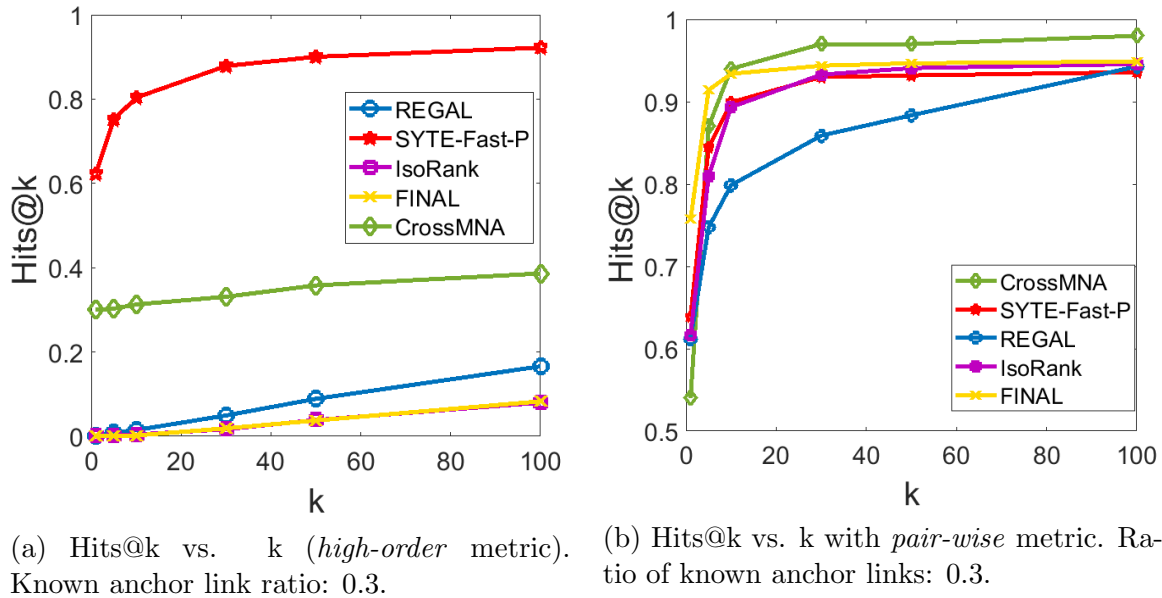


Figure 5.2: Cross-network node retrieval results on *DBLP* dataset. Hits@k vs. k. Higher is better. Best viewed in color.

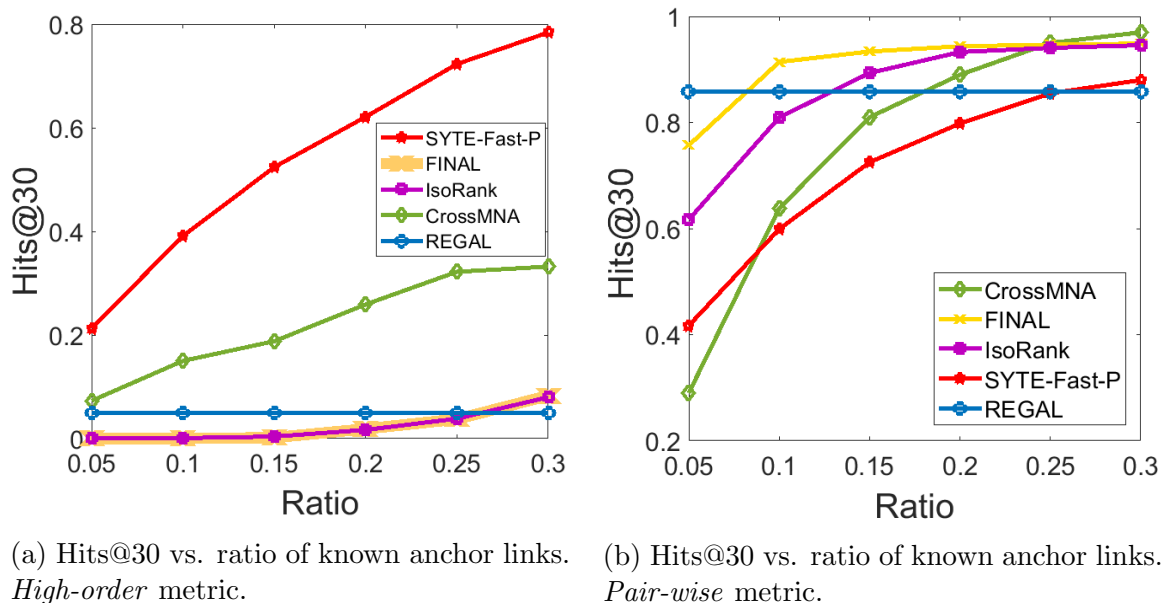


Figure 5.3: Cross-network node retrieval results on *DBLP* dataset. Hits@30 vs. ratio of known anchor links. Higher is better. Best viewed in color.⁶

5.4.2 Effectiveness Results

First, we present the experimental results of the multi-network alignment task. We focus on one-to-one alignment in this experiment. In order to obtain the one-to-one mapping, we implement a high-order greedy match algorithm to convert the multi-way association solution tensor \mathcal{X} to a matching tensor \mathcal{M} , in which each fiber (e.g. $\mathcal{M}(i_K, i_{K-1}, \dots, i_2, :)$) contains at most one non-zero entry to indicate a one-to-one alignment.

We use two metrics to evaluate the effectiveness. First, for a given one-to-one alignment tuple of nodes (e.g. (u_1, \dots, u_K) , $u_1 \in G_1, \dots, u_K \in G_k$), we consider it as a successful alignment iff *all* nodes in the tuple are correct (referred to as the *high-order metric*). For the second metric, for a given one-to-one alignment tuple of nodes, we consider it successful iff *any* pair of nodes (e.g. u_1 and u_2) in the tuple are aligned correctly (referred to as *pair-wise metric*). For both metrics, the alignment accuracy is calculated as $\frac{\# \text{ of correctly aligned node tuples}}{\# \text{ of node tuples in test data}}$, where the test data does not contain any known multi-way anchor links. The results on *Arxiv* without attribute are shown in Figure 5.4. We observe that by the high-order metric, both basic algorithm and SYTE-Fast-P algorithm outperform baselines by up to 16.2%. By the pair-wise metric, our proposed methods cannot outperform, but are comparable with baselines. This is consistent with the goal of the proposed SYTE methods which are designed to primarily capture multi-way (i.e., high-order) associations. On the other hand, the baseline methods, being pair-wise approaches, are better suited for pair-wise association inference.

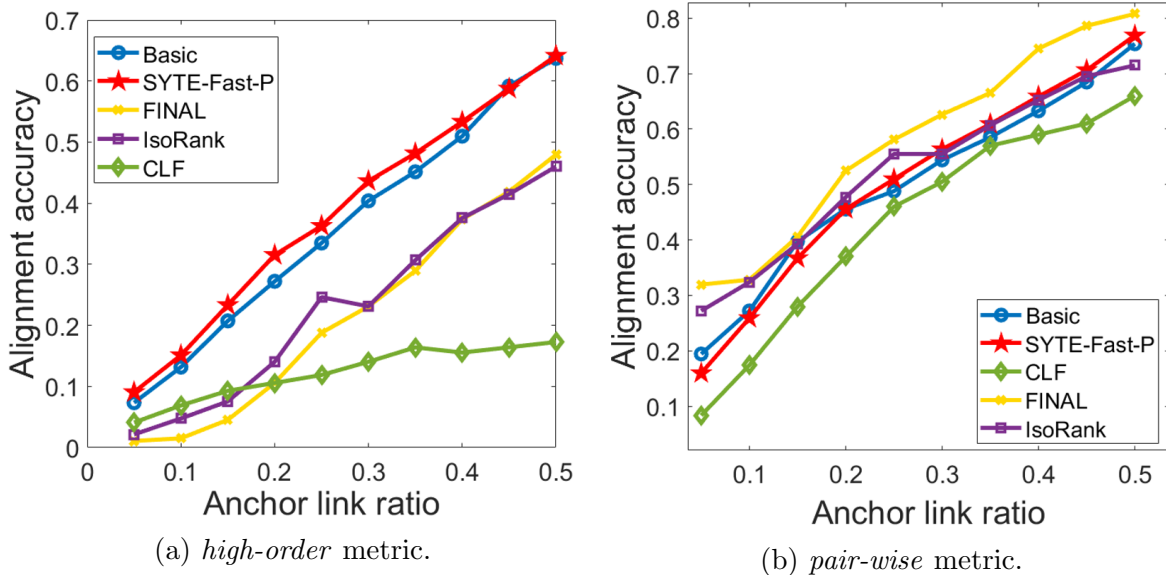


Figure 5.4: Multi-network alignment results on *Arxiv* dataset (without node attributes). Best viewed in color.

⁶Note that the curves of *REGAL* in both (a) and (b) are flat because *REGAL* is an unsupervised method.

The results of multi-network alignment of attributed networks on *DBLP* dataset are presented in Figure 5.5. In most cases, both basic algorithm and SYTE-Fast-A outperform baseline methods. Although there are some performance loss of SYTE-Fast-A compared to the basic algorithm, the alignment accuracy of SYTE-Fast-A is still higher than baseline pair-wise network alignment methods.

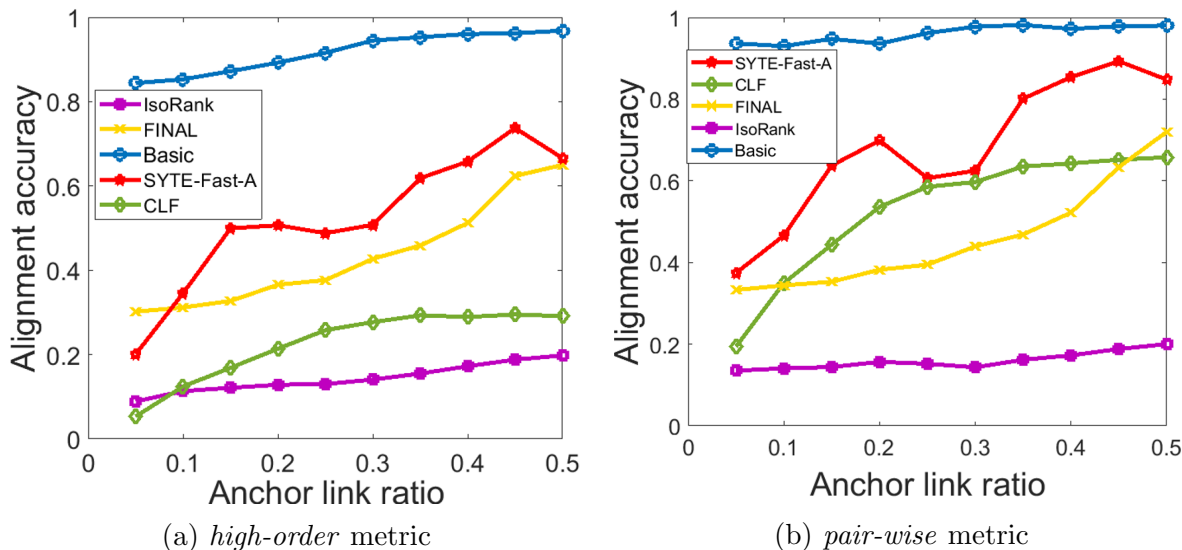


Figure 5.5: Multi-network alignment results on *DBLP* (with attributes). Best viewed in color.

Second, we present the experimental results of the high-order recommendation task. We focus on one-class recommendation in this task [171]. On one hand, for each user in the test data, if the returned top- k tuple list of (artist, tag) contains the ground-truth, we consider it as a successful hit. Similar to multi-network alignment, this is referred to as *high-order* metric. On the other hand, if either artist *or* tag is correctly recommended to a given user, we consider it as a successful recommendation. This is referred to as *pair-wise* metric. We use hits@30 for both metrics. The results are shown in Figure 5.6. We can observe that the proposed algorithm SYTE-Fast-P outperforms baselines in terms of both *high-order* and *pair-wise* metrics.

Next, we present the experimental results of the multi-network node retrieval task. For each query node of a given network, the task retrieves nodes from the other two networks for a top- k list. We study the hits@ k vs. k for a fixed ratio of known anchor multi-way associations, and then fix $k = 30$ to study the hits@30 vs. the ratio of known multi-way anchor links. *DBLP* dataset is used for this task, and the results are shown in Figure 5.2 and Figure 5.3. In Figure 5.2 (a) and Figure 5.3 (a), we can see that SYTE-Fast-P outperforms baselines by a large margin (e.g., by 50%+ when $k = 100$). In Figure 5.2 (b) and Figure

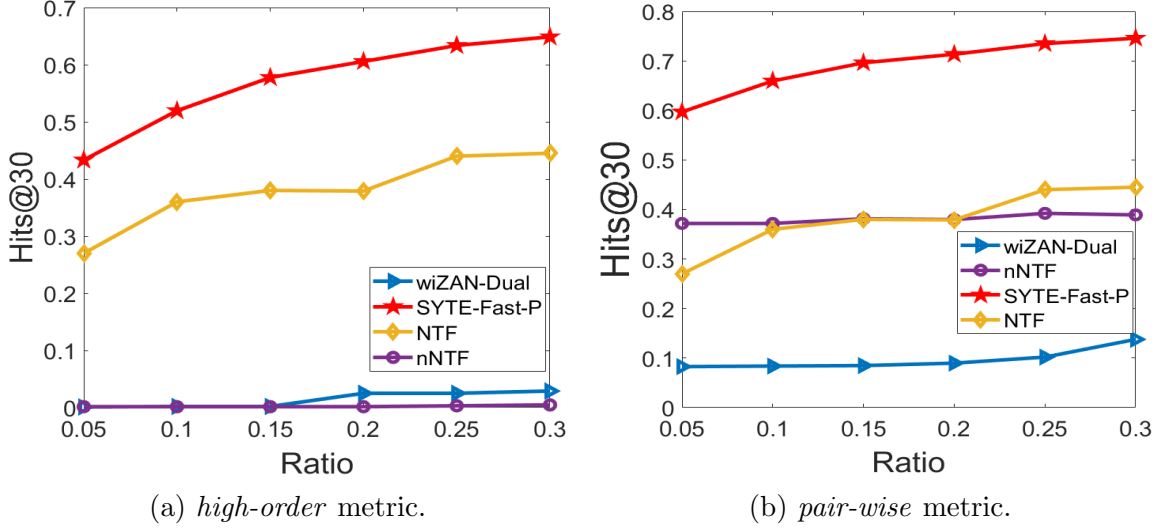


Figure 5.6: High-order recommendation results on *LastFm* dataset. Best viewed in color.

5.3 (b), the proposed SYTE-Fast-P algorithm does not outperform some pair-wise network alignment methods (e.g., *CrossMNA*, *FINAL*). This is also expected since the pair-wise metric is used for the retrieval task, whereas the proposed SYTE algorithms are primarily designed for the high-order metric (i.e., multi-way association).

5.4.3 Scalability Results

In the heart of our proposed algorithm is a Sylvester equation solver. Here, so we compare the proposed methods with two classic Sylvester equation/linear system solvers, i.e., *Fixed Point method (FP)* and *Conjugate Gradient method (CG)*⁷. We extract subgraphs from the largest dataset *Aminer*, and the results are presented in Figure 5.7. We terminate the program if it can not finish in 3,000 seconds. Note that the vertical axes are in *log* scale. From Figure 5.7 (a), for plain networks, our proposed method SYTE-Fast-P exhibits a *linear* scalability w.r.t. the number of nodes of the input networks, whereas neither of the two baseline methods (*FP* and *CG*) can finish within 3,000 seconds with more than 1,200 nodes. SYTE-Fast-P* is a variant of SYTE-Fast-P, and it is detailed in Algorithm 5.3. SYTE-Fast-A* is a variant of SYTE-Fast-A, which uses SYTE-Fast-P* in line 4 of Algorithm 5.4. From Figure 5.7 (b), the proposed SYTE-Fast-A scales linearly whereas all other methods scales super-linearly. Note that the blue curve (marked as SYTE-BCD) denotes a variant of SYTE-Fast-A by using the exact block coordinate descent method. As

⁷Baseline methods *FINAL* and *IsoRank* also uses *FP* method. The supervised learning methods (e.g. *CrossMNA*) are difficult to be compared with our numerical method since they require off-line training.

we can see, it can not scale up to large networks.

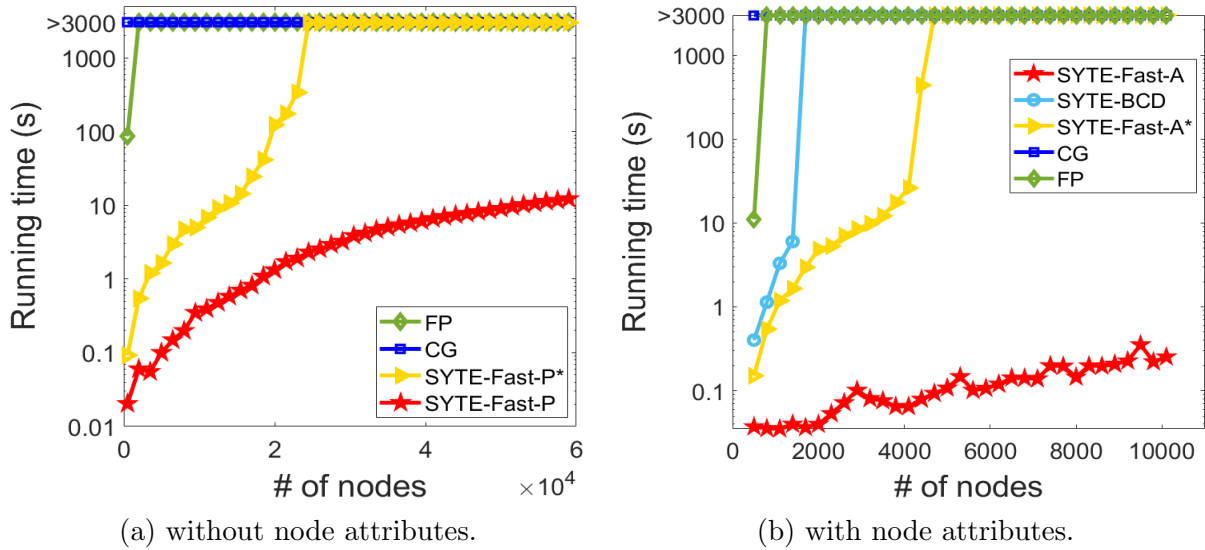


Figure 5.7: Scalability results and running time comparison on *Aminer* dataset. Notice the *log* scale in the vertical axis.

5.4.4 Parameter Sensitivity

Here, we study the parameter sensitivity of the proposed algorithm SYTE-Fast-P. We use the multi-network alignment task to study the alignment accuracy w.r.t. two key parameters (i.e., α and Krylov subspace dimension l for SYTE-Fast-P). We use three subgraphs extracted from *Douban* dataset. The results are shown in Figure 5.8. From Figure 5.8, we can see that the performance of Algorithm 5.2 is stable in a relatively large range of parameter space.

The parameter sensitivity study of SYTE-Fast-P* is presented in Figure 5.9. The performance of SYTE-Fast-P* increases with the rank of eigen-decomposition, and α has small impact on the performance since we use uniform multi-way anchor link tensor here. As we can see, compared with Figure 5.8, SYTE-Fast-P is relatively more stable w.r.t. the parameters α and subspace size, while the rank of eigen-decomposition has higher impact on the performance of SYTE-Fast-P*. The scalability results on the number of networks are shown in Figure 5.10. The number of nodes used for each network is 100, and when the running time is larger than 3,000s, the program is terminated. The vertical axis is in *log* scale. As we can see, the proposed methods show exponential scalability w.r.t. the number of networks (relatively much smaller than the number of nodes) as we analyze in Section 5.2.3 and 5.3.

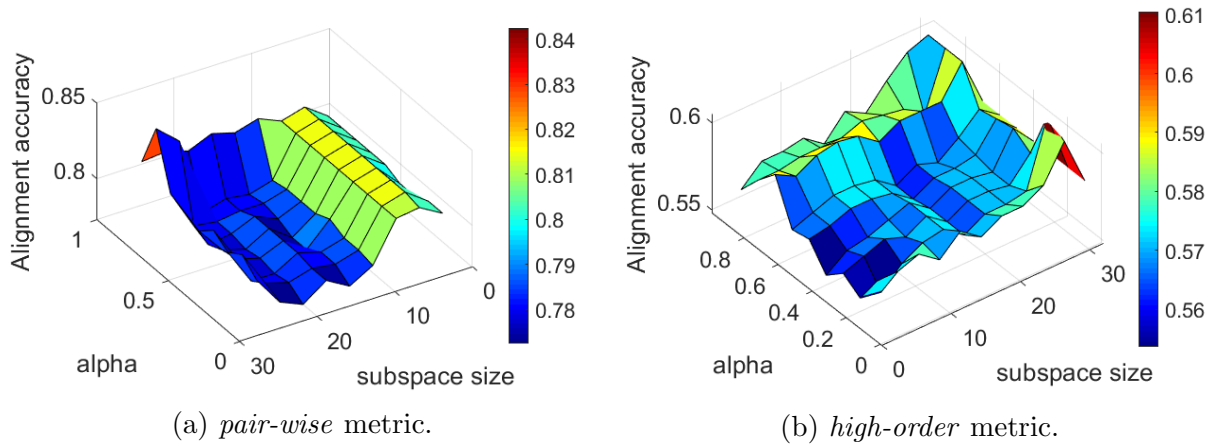


Figure 5.8: Parameter sensitivity analysis of the proposed Algorithm 5.2.

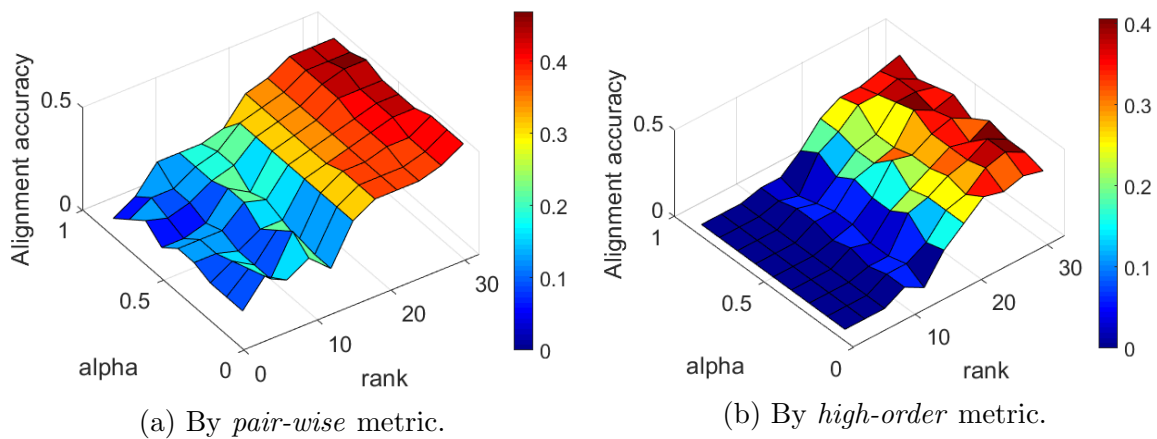


Figure 5.9: Parameter sensitivity of SYTE-Fast-P* on *Arxiv*.

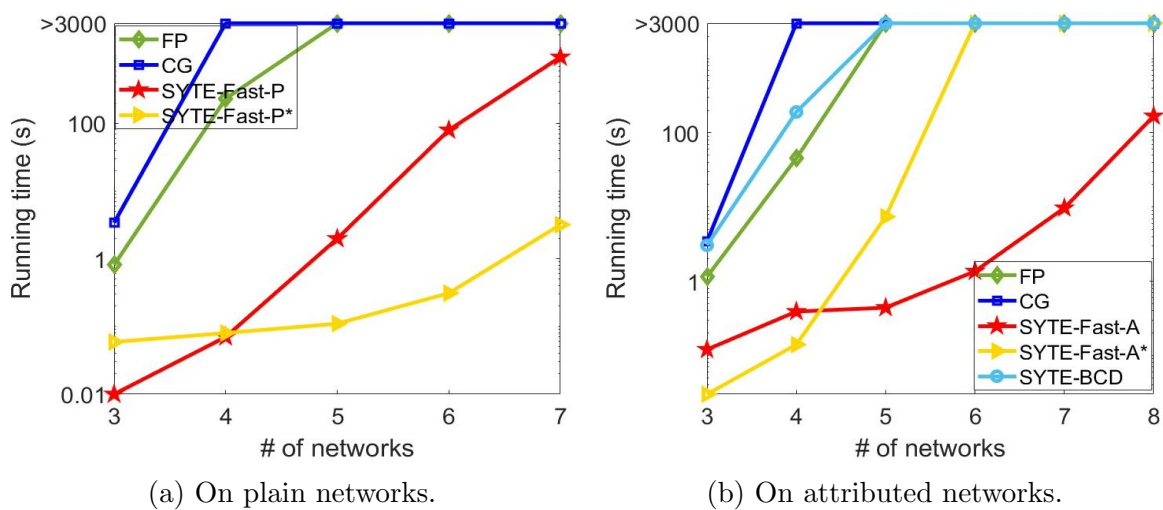


Figure 5.10: Scalability results of running time vs. the number of networks on *DBLP* dataset.

CHAPTER 6: HIGH-ORDER ASSOCIATION WITH NEURAL TECHNIQUES

In this chapter, we present our works for high-order multi-network association using neural techniques. Specifically, we first introduce a multi-resolution multi-network embedding framework, which aims at learning the embeddings of network elements of different resolutions and from different networks into the same embedding space in an unsupervised manner [26]. Next, we describe our work in hypergraph representation learning, with the assistance from pre-training strategy [29]. Lastly, we will present a case study of applying the hypergraph representation learning framework on a real-world application for online shopping platforms.

6.1 MULTI-RESOLUTION MULTI-NETWORK EMBEDDING

Network embedding is the cornerstone of many real-world applications, and has been receiving much research attention in recent years. It offers a powerful way to encode the underlying network characteristics (e.g., topology, attribute) into a compact low-dimensional space. As such, it has benefited a variety of downstream data mining tasks (e.g., node/network classification, link prediction, and clustering), often with a significantly boosted empirical performance. Despite much progress has been made, most of the previous work has not adequately, if at all, addressed two fundamental limitations, which we elaborate below.

First, Most of the previous work, with only a few exceptions, focuses on a *single network*. For multiple input networks, these methods will learn embeddings of different networks separately, and thus might result in disparate embedding space. To see this, let us take node embedding as an example. A dominant branch of node embedding (e.g., *Deepwalk*, *LINEM* and many of their follow-ups [20, 172, 173, 174]) relies on identifying appropriate node proximity/context based on truncated/short random walks. The identified node context will be then preserved in the embedding space, often through a language model (e.g., Continuous Bag of Words (CBOW) and SkipGram). However, any node pair across different networks are disconnected without auxiliary information (e.g., anchor links). In other words, nodes in one network will never be the context of nodes from another network and vice versa. Therefore, the node embeddings of different networks will be in different or disconnected space. This would render the inapplicability of some downstream mining tasks (e.g., cross-layer dependence inference in multi-layered network systems [175]) or add extra complexity for other mining tasks. For example, with the node embeddings from such methods as inputs, we would have to train an additional classifier for network alignment task, constrained by

the availability of extra anchor links.

Second, most, if not all, of the previous work is designed to learn embeddings at *single resolution*. For example, the vast majority of network embedding focuses on *node* embedding (e.g., *node2vec* [172], *LINE* [174], *Deepwalk* [173] and many more); at the coarser resolution, *subgraph2vec* [176] and *deep graph kernel* [177] learn the embeddings of subgraphs; at the coarsest resolution, *graph2vec* [178] focuses on learning vector representations for the entire networks. To the best of our knowledge, almost none of the previous methods support multi-resolution network embedding, although objects (e.g., nodes, subgraphs, and networks) across different resolutions are intrinsically correlated with each other. For example, networks with similar subgraph distributions tend to be similar to each other [177]; [20] indicates that, nodes inside similar subgraphs are likely to belong to the same category. Ignoring such cross-resolution correlation during the embedding learning process is likely to lead to sub-optimal results. On the contrary, if we could have objects at different resolutions in the same embedding space, it might greatly benefit certain downstream applications. For example, for network science of teams [132], by embedding both team members (nodes of the underlying person network) and teams (subgraphs) in the same space, it would immediately enable effective team member recommendation by calculating the similarity between a candidate team member and a given target team, say based on the cosine similarity between the member embedding and team embedding.

In order to address the above limitations, we propose a unified method (MRMINE) to learn the multi-resolution multi-network representation simultaneously in a mutually beneficial way. The key idea is to construct the cross-resolution cross-network context for each object of each network at three complementary resolutions, including node, subgraph and network. The constructed corpus of such network context can be then fed into a variety of language models, such as CBOw, SkipGram, etc., to generate the embeddings for different network objects (nodes, subgraphs, networks) from different networks in the same space. The proposed method is highly efficient and scalable, with a time complexity of $O(Hn\log(n))$ (where n is the number of nodes of input networks and H is a constant much smaller than n) for the basic version. We further propose an accelerated version with a *linear* time complexity w.r.t. the number of nodes and edges of input networks.

Figure 6.1 presents an illustrative example of the simplified procedure of multi-resolution and multi-network embedding. Figure 6.1 (a) shows three input networks selected from bioinformatics dataset. (b1) represents the Cross-Resolution Cross-Network (CRCN) relation network in which we connect vertices of all three types of objects (i.e. nodes, subgraphs, and networks, represented as circles, squares, and hexagons respectively) in the same network, according to the node-subgraph, subgraph-network membership (vertical black solid

links) and subgraph similarities (horizontal red dashed links). For example, enzyme #33 contains a subgraph (we use Weisfeiler-Lehman (WL) subtree [179] in this example) numbered in #31, which contains node #16 and node #17. So the blue hexagon is connected to the gray square #31, and the gray square #31 is connected to the blue circle #16 and circle #17. Subgraph #31 and subgraph #34 are structurally similar with only one node difference (Figure (b2)), so gray squares #31 and #34 are connected. Context of objects from different resolutions and networks are then extracted from the CRCN relation network to learn the embeddings in (c). As we can observe from (c), the green network is close to the blue network (two enzymes of the same category) while both are far apart from the orange network (a mutag instance). Node #10 and #11 are close because they are connected by the same square node in (b1), which means they are rooted at the same WL subtree. Subgraph #31 and subgraph #34 are close in the embedding space although they exist in different networks. Also from different networks, node #16 and node #21 are close because they are structurally similar (connected to similar subgraph #31 and subgraph #34 respectively). Our method embeds multi-resolution multi-network objects into the same space, and it naturally enables downstream multi-network mining tasks.

The main contributions of this section are:

- **Problems.** To the best of our knowledge, we are the first to study the problem of learning multi-resolution multi-network embeddings.
- **Algorithms.** We propose effective and efficient algorithms for learning the embeddings of multi-resolution and multi-network objects simultaneously.
- **Empirical Evaluations.** We perform extensive experimental evaluations on a diverse set of real networks, which demonstrate that our methods (1) enable and enhance a variety of graph mining tasks, such as collective network alignment, and (2) scale up to million-node graphs.

6.1.1 Problem Definition

In this section we formally define the problem of multi-resolution multi-network embedding. The symbols and notations used in this paper are summarized in Table 6.1. Let $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$ represent a set of k input networks, and $\mathcal{S} = \{S_1, S_2, \dots, S_l\}$ represents the subgraph set which contains all the subgraphs extracted from each graph in \mathcal{G} of a particular type. For example, \mathcal{S} could be a set of all Breath First Search (BFS) subtrees or Weisfeiler-Lehman (WL) subtrees with height less than 3 exacted from all networks in \mathcal{G} . In

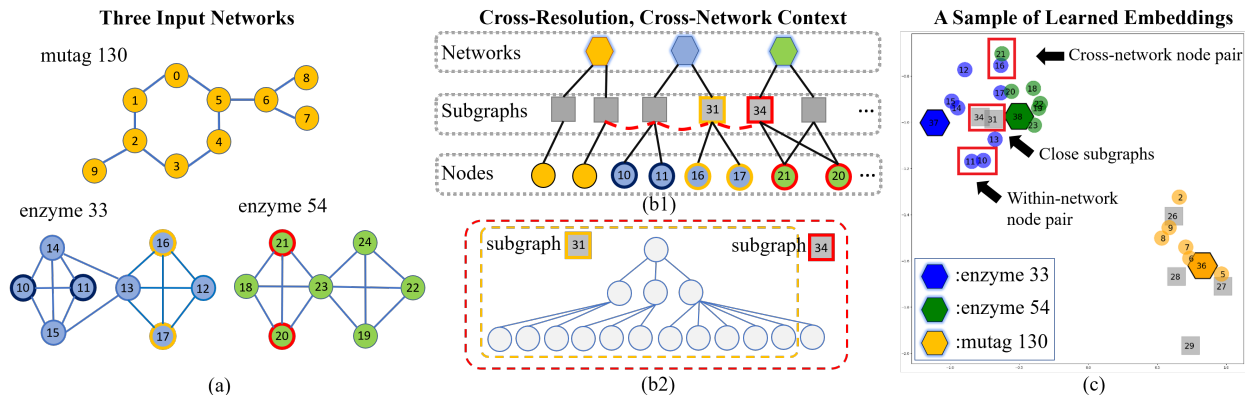


Figure 6.1: an illustrative example of multi-resolution multi-network embedding. (a) shows three small molecular graphs from bioinformatics dataset. The two lower graphs belong to the same category of enzyme. (b1) shows the Cross-Resolution Cross-Network (CRCN) relation network constructed by our method. Some subgraphs/nodes are omitted for brevity. (b2) shows two similar subgraphs (WL subtrees here) extracted from enzyme #33 and enzyme #54, numbered in #31 and #34. (c) shows the learned 2-d representation for all network objects. Some embeddings are omitted to avoid overlap.

this paper, we use WL subtrees for all the algorithms. With these notations, the problem of multi-resolution multi-network embedding can be defined as follows.

Problem 6.1. MULTI-RESOLUTION MULTI-NETWORK EMBEDDING

Given: (a) the inputs for constructing Cross-Resolution Cross-Network (CRCN) relation network: (a1) a set of networks \mathcal{G} , (a2) the dimension of embedding vectors p ; (a3) the maximum height of WL subtrees H ; and (b) the parameter set for corpus generation and SkipGram model (e.g. SkipGram window size w , random walk length l , etc.).

Output: the embedding matrices \mathbf{F}_g , \mathbf{F}_s , and \mathbf{F}_n for (1) all input networks in \mathcal{G} , (2) all extracted subgraphs in \mathcal{S} , and (3) all nodes in \mathcal{G} , respectively, with all embeddings in the same space.

We adopt the following terminologies and notations for simplicity of algorithm description. First, We use *vertex* and *vertices* to indicate the nodes in the CRCN relation network, and *nodes* to only indicate nodes in the original network set. Second, we use function p_n, p_s, p_g for the mappings from original network object to the vertices in the CRCN relation network, and q_n, q_s, q_g for the mapping from vertices in the relation network back to original network objects. For example, $p_s(L_G^n) = v$ maps the subgraph with label L_G^n (meaning WL subtree rooted at node $n \in G$; every unique WL subtree can be represented by a distinct label) to vertex v in the CRCN relation network, and $q_s(v) = l$ maps the vertex v from the CRCN relation network back to a subgraph label l . Similarly, p_n, q_n are used for node level mapping

Table 6.1: Major Notations and Definition

Symbols	Definition
$\mathcal{G} = \{G_1, G_2, \dots, G_k\}$	a set of k graphs
$\mathcal{R}, \mathcal{E}_R$	cross-resolution cross-network relation network and edge set
\mathcal{E}_{R_s}	a set of edges within subgraph vertices
\mathcal{E}_{R_g}	a set of edges between subgraph vertices and network vertices
\mathcal{E}_{R_n}	a set of edges between subgraph vertices and node vertices
$\mathcal{L} = \{L_{G_1}^{n_1}, \dots, L_{G_k}^{n_k}\}$	a set of multi-set labels for all WL subtrees in the graph set \mathcal{G}
$\mathbf{F}_g, \mathbf{F}_s, \mathbf{F}_n$	embedding matrices for networks, subgraphs, and nodes
H	the maximum height of WL subtrees
L^n / L_G^n	the WL subtree label of node n /specifically $n \in G$
$(L_G^n)_i$	the WL label of node $n \in G$ at i -th WL iteration
Q_{S_i}, Q_{S_j}	the sorted degree sequence of subtree S_i, S_j
p	the dimension of embedding vectors
$\Phi(n)$	the embedding of vertex n
p_n, p_s, p_g	mapping functions from original objects to CRCN network
q_n, q_s, q_g	mapping functions from CRCN network to original objects

and p_g, q_g are used for network level mapping.

To illustrate the intuition of Problem 6.1, let us make an analogy between multi-resolution multi-network embedding and text embedding. The objects of multiple networks, subgraphs, and nodes in our problem setting can be seen as documents, sentences, and words in text embedding respectively. In this case, the problem studied by [180] embeds sentences/paragraphs with words in a document, which resembles our multi-resolution problem setting.

6.1.2 Proposed Methods: MRMINE

In this section, we introduce our proposed method. We start with the preliminaries followed by challenges and key ideas. We then present the detailed demonstration of our basic algorithm and the accelerated algorithm.

Preliminaries.

Weisfeiler-Lehman (WL) subtree. WL subtree [179] is a subgraph with tree structure rooted at a designated node in a network. The height of the WL subtree is the maximum distance between the root node and any other nodes. Unlike the BFS subtree, the WL subtree treats the repetition of nodes in the node search of tree construction as distinct nodes. For example, in the upper right WL subtree in Figure 6.2, node #0 appears in both level 0 and level 2, because node #0 is the neighbor of node #1, #3 and #4 in level 1, and in level 2 it is regarded as a distinct node.

WL label transformation. Originated from the Weisfeiler-Lehman graph isomorphism test,

the WL label transformation iteration [179] is an algorithm used for relabeling node labels. The time complexity is $O(hm)$, where h is the iteration number/height of WL subtrees, and m is the number of edges. The multi-set labels generated by the i -th WL iteration can be regarded as the identity of unique WL subtrees with height of i [176]. In each iteration of WL label transformation, a new set of labels is produced for all nodes in a network. Since each unique label corresponds to a particular WL subtree, we use the labels (e.g. L_G^n) as IDs for subtrees in this paper. Therefore, it can be used for efficiently generating subgraphs on each node, and building the vocabulary of subgraphs.

Challenges and Key Ideas. In order to learn the multi-resolution multi-network embedding on the same space, there are several challenges as follows. First, how to build cross-resolution cross-network context, so that objects at different resolutions from different networks could be in each other’s context, which would in turn allow to find their embeddings in the same space? Our first key idea is to introduce Cross-Resolution Cross-Network (CRCN) relation network with vertices representing multi-resolution multi-network objects, and edges representing cross-resolution cross-network relations between objects. For example, in Figure 6.1 (b1), the CRCN relation network aims to capture the structural relationship among nodes, subgraphs, and three molecular networks. Second, how to construct the links of CRCN relation network? Our second key idea is to use WL subtrees for the subgraph resolution. It bears three advantages as follows. (1) the WL label transformation algorithm can be used to efficiently generate WL subtrees as subgraphs [179]; (2) the WL subtrees can act as bridges to build links between network/subgraph/node vertices (i.e. cross-resolution links); (3) the ’borderless’ WL subtrees in conjunction with the similarity defined over subgraphs help to build cross-network links. Third, how to reduce the computation cost to build CRCN relation network (e.g. the number of subgraphs could be large, and it could be time-consuming to build subgraph-subgraph similarity links, etc.)? Our third idea is to explore a hierarchical structure of WL subtrees for the subgraph resolution in the CRCN relation network construction, for the sake of avoiding explicit cross-network links (e.g. red dashed links in Figure 6.1 (b1)), but meanwhile still preserving the cross-network subgraph similarities.

Basic MrMine Method. First, we adopt H iterations of WL node label transformation to generate unique WL subtrees of up to height H [176, 179]. Specifically, in order to construct the cross-network links between subgraphs, we propose to use a function $f(S_i, S_j)$ to calculate the similarity between subgraph S_i and S_j . Similar to the selection of subtrees, the selection of function f is also not limited. The most intuitive method of calculating similarities between two graphs is using graph kernels. In this case, $f(S_i, S_j) = K(S_i, S_j)$, in which K can be any graph kernels such as random walk graph kernel [117], WL subtree

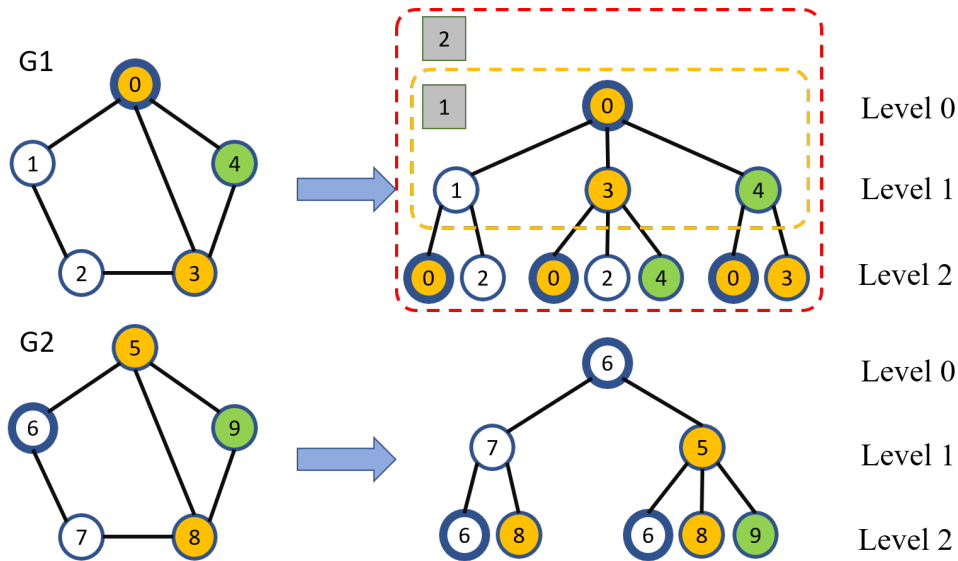


Figure 6.2: an illustrative example of WL subtree. The right trees are 2-level WL trees rooted at node #0 and #6, respectively. Best viewed in color.

kernel [179], etc. Although in common case, calculating graph kernel is costly ($O(n^3)$ in which n is the number of nodes [117]) without approximation, subgraphs are smaller-scaled (compared to networks) and can be calculated relatively efficient. We provide two more efficient f functions below.

We observe that, for WL subtrees, the structural characteristic is essentially preserved by the node degrees of each level. For example, in Figure 6.2, the structural characteristic of 2-level subtree can be simply represented by the degree sequence of node 0 concatenated with the sorted degrees of its neighboring node 1, 3, and 4, which gives level 1: 3, level 2: 2,2,3. Since node 3 is structurally equal to node #0 (belonging to isomorphic WL subtree), they both produce the same degree sequence. The degree sequence of WL subtree rooted at node #1 (i.e. level 1: 2, level 2: 2,3) is distinct from that of the WL subtree rooted at node #0 and #3. Therefore the subtree structures are also quite different. We adopt Dynamic Time Wrapping (DTW) [20] to measure the distance between two degree sequences at each level, and connect subgraphs with structural scores lower than a threshold. Formally, for sorted degree sequence Q_{S_i} and Q_{S_j} of subtrees S_i and S_j with length l_{S_i} and l_{S_j} :

$$f(Q_{S_i}, Q_{S_j}) = \sum_h DTW(Q_{S_i}^h, Q_{S_j}^h) \quad (6.1)$$

where $Q_{S_i}^h, Q_{S_j}^h$ are the sorted degree sequences of S_i and S_j at level h , respectively. However, DTW might fail to distinguish between high-level degree sequences of different length. For example, the sorted degree sequence for 2-level subtree of node 0 in G_1 (which is 2, 2, 3)

has zero DTW value compared with the same level degree sequence of node 6 in G_2 (which is 2, 3). To address this issue, we propose to use a method similar to Spearman’s footrule distance, which is commonly used in ranking list comparison.

$$f(Q_{S_i}, Q_{S_j}) = \sum_h \sum_t |\tilde{Q}_{S_i}^h(t) - \tilde{Q}_{S_j}^h(t)| \quad (6.2)$$

where $\tilde{Q}_{S_i}^h$ is the sorted degree sequence on level h after filling zeros to the front of the original list, $Q_{S_i}^h$, to make $\tilde{Q}_{S_i}^h$ and $\tilde{Q}_{S_j}^h$ have the same length (i.e. $\max(l_{S_i}, l_{S_j})$). $t = 1, \dots, \max(l_{S_i}, l_{S_j})$.

Algorithm 6.1 CRCN Relation Network Builder

Input: Given input network set $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$, maximum subtree height H .

Output: The CRCN relation network \mathcal{R} .

- 1: **for** $G \in \mathcal{G}$ **do**
 - 2: Conduct H WL relabeling iterations to generate multi-set labels \mathcal{L} .
 - 3: **end for**
 - 4: Set edge set of \mathcal{R} : $\mathcal{E}_R = \Phi$.
 - 5: **for** each label $L_G^n \in \mathcal{L}$ **do**
 - 6: **if** $(p_s(L_G^n), p_n(n)) \notin \mathcal{E}_R$ **then**
 - 7: Add edge $(p_s(L_G^n), p_n(n))$ to \mathcal{E}_R .
 - 8: **end if**
 - 9: **if** $(p_g(G), p_s(L_G^n)) \notin \mathcal{E}_R$ **then**
 - 10: Add edge $(p_g(G), p_s(L_G^n))$ to \mathcal{E}_R .
 - 11: **end if**
 - 12: **end for**
 - 13: Return the CRCN relation network \mathcal{R}
-

The proposed CRCN relation network building algorithms are summarized in Algorithm 6.1 and Algorithm 6.2. Algorithm 6.1 builds the CRCN relation network without cross-network edges. Algorithm 6.2 generates the cross-network edges between subgraph vertices.

In line 2 of Algorithm 6.1, we conduct H WL relabeling iterations to generate multi-set node labels for the IDs of WL subtrees ($O(Hm)$). From line 5 to line 12 we add edges between node vertices and subgraph vertices, and edges between network vertices and subgraph vertices ($O(Hkn)$). Note that no edges between node vertices are added although edges exist between nodes in the original network.

After the CRCN relation network \mathcal{R} is constructed, we conduct truncated random walk [173] on each vertex of \mathcal{R} to build the corpus of multi-resolution multi-network objects,

Algorithm 6.2 Cross-Network Subgraph Context Builder

Input: Given a multi-resolution relation network \mathcal{R} (with subtree vertex list V and edge set \mathcal{E}_R), threshold σ , window size w .

Output: The multi-resolution relation network with added cross-network links of subgraph vertices.

- 1: Sort V of subtree vertices by the summation of degree sequence.
 - 2: **for** each $v \in V$ **do**
 - 3: **for** each subtree s in the window of size w of v **do**
 - 4: **if** $f(Q_s^h, Q_v^h) < \sigma$ **then**
 - 5: Add edge $(p_s(v), p_s(s))$ to \mathcal{E}_R
 - 6: **end if**
 - 7: **end for**
 - 8: **end for**
 - 9: Return \mathcal{R}
-

which is similar to build corpus for nodes in a single network. The corpus can be further fit into a language model such as SkipGram with negative sampling or hierarchical softmax techniques. Here we use SkipGram with negative sampling. The overall model is summarized in Algorithm 6.3. Line 1 and 2 use Algorithm 6.1 and Algorithm 6.2 to construct the multi-

Algorithm 6.3 MRMine

Input: Given a set of networks $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$, the height of WL subtrees H , the embedding dimension p , the window size w_1, w_2 for adding cross-network edges of subgraph vertices and SkipGram model respectively, and the threshold σ .

Output: The network embedding matrix \mathbf{F}_g for \mathcal{G} , the subgraph embedding matrix \mathbf{F}_s , and the node embedding matrix \mathbf{F}_n .

- 1: Construct multi-resolution relation network \mathcal{R} by Algorithm 6.1.
 - 2: Update \mathcal{R} by Algorithm 6.2.
 - 3: Construct corpus W by applying truncated random walk on each vertex in \mathcal{R} .
 - 4: **for** vertex u_i in each random walks $r \in W$ **do**
 - 5: $J(\Phi) = -\log Pr(u_j | \Phi(u_i)), u_j \in r[i - w_2, i + w_2]$
 - 6: $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$
 - 7: **end for**
 - 8: Return embedding matrices $\mathbf{F}_n, \mathbf{F}_s, \mathbf{F}_g$.
-

resolution multi-network relation network for each network object. Line 3 builds corpus that preserves the structural information of the relation network. Line 4 to line 7 are SkipGram model that learns the embeddings of nodes, subgraphs, and networks simultaneously.

Complexity Analysis on MrMine. In Algorithm 6.2, instead of comparing every pair of subgraphs for adding cross-network links for subgraph vertices (with time complexity $O(H^2n^2)$), line 1 generates a sorted list of subtrees with the summation of their corresponding

degree sequence for line 3 to only compares the structural similarity of the target vertex with other vertices within the window of size w in the list. Since sorting the list V only cost $O(n \log n)$, if the window size w is bounded by $O(n \log n)$, this method can lower the time complexity of Algorithm 6.2 from $O(H^2 n^2)$ to $O(Hn \log(n))$. Line 4 can use either Eq. (6.1) or (6.2).

In Algorithm 6.3, since the constructed CRCN relation network has at most $(Hk + k)n + k$ vertices (linear w.r.t. n), and all the steps in MRMine have linear complexity except using Algorithm 6.1 in line 2, the overall time complexity of MRMine is $O(Hn \log(n))$ (assuming that n and m have the same order of magnitude).

Accelerated MrMine+ Method. The base MrMine model introduces both cross-resolution and cross-network links in the CRCN relation network to capture the network objects' relation both across different layers of resolutions and different networks. To further reduce the time complexity of MRMine, we propose the improved model MRMine+. The most intuitive way is to simply remove the cross-network edges between subgraph vertices (Algorithm 6.2), which immediately reduces the time complexity from $O(Hn \log n)$ to $O(Hm + Hkn)$. However, this method would not only reduce the ability of preserving cross-network similarities of the CRCN relation network, but also might lead to disconnected CRCN relation network. Instead, we explore the hierarchical structure of WL subtrees, and propose a hierarchical CRCN (H-CRCN) relation network. The idea is to preserve the structural characteristics across networks from different subgraph granularities (e.g. Figure 6.3). The advantages are two-fold: (1) it largely reduces the time complexity of CRCN relation network construction; and (2) still preserves the cross-network context information without explicitly generating cross-network links. The details are as follows. First, we construct a reversed hierarchical subgraph tree. The root of the tree is 0-level WL tree (i.e. node), and the vertices in level i represents the WL-subtrees of height i . Two vertices are connected if the WL subtree corresponding to a vertex from the higher level can be generated from a vertex's WL subtree of lower level. For example, in Figure 6.2, subtree 2 (level 2) can be generated by subtree 1 (level 1) by applying one more iteration of WL relabeling iteration, so subgraph vertex 1 and 2 are connected in Figure 6.3 (gray square 1 and 2). Next, network vertices or node vertices are connected to the last level of subgraph tree, based on the membership relation between the networks/nodes and the last level of WL subtrees. For example, in Figure 6.3 (b), node 0, 3, 5, 8 are connected to subtree vertex 2 because all these nodes can generate WL subtree 2 at level 2. The complete model is summarized in Algorithm 6.4. We use \mathcal{E}_{R_s} to indicate edge set within the reversed relation tree of WL subtree vertices (e.g. edges within the dashed rectangle in Figure 6.3), \mathcal{E}_{R_g} to indicate edges between network vertices and the highest level of WL subtree vertices (e.g. edges between hexagons and squares

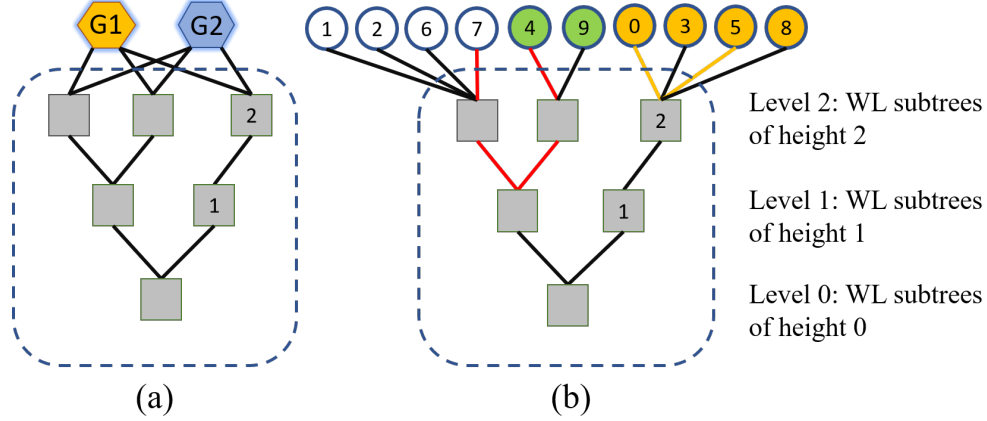


Figure 6.3: an example of H-CRCN relation network structure constructed from Figure 6.2. The reversed gray trees in the dashed rectangle are the hierarchical relation network of WL subtrees. (a) captures the relation of two networks (hexagons) with this hierarchical subgraph tree; (b) captures the relation of nodes (circles) with this hierarchical subgraph tree. Best viewed in color.

in Figure 6.3 (a)), and \mathcal{E}_{R_n} to indicate edges between node vertices and the highest level of WL subtree vertices (e.g. edges between circles and squares in Figure 6.3 (b)).

With such a way of constructing H-CRCN relation network, similarities of network objects are captured in different subgraph granularities. For example, in Figure 6.3, node 0 and 5 are connected by the yellow short path while node 4 and 7 are connected by the red long path, which indicates that node 0 and 5 are closer (both connected to a subgraph of finer granularity).

In Algorithm 6.4, the \mathcal{E}_{R_s} is constructed from line 5 to line 9. \mathcal{E}_{R_g} and \mathcal{E}_{R_n} are constructed from line 10 to line 13. Note that \mathcal{E}_{R_g} and \mathcal{E}_{R_n} are generated independently by the edges of (subgraph vertex, network vertex) and the edges of (subgraph vertex, network vertex) respectively. From line 14 to 16, we apply truncated random walks for building the corpus P_s , P_g and P_n with only \mathcal{E}_{R_s} , $\mathcal{E}_{R_s} \cup \mathcal{E}_{R_n}$ and $\mathcal{E}_{R_s} \cup \mathcal{E}_{R_n}$, respectively. The SkipGram model is applied on the union of P_g , P_n and P_s from line 17 to line 21.

Complexity Analysis on MrMine+. H iterations of WL relabeling processes cost $O(Hm)$. From line 5 to line 9, with the absence of cross-network edges between subgraph vertices, only subgraph vertices need to be traversed once, with the complexity of $O(Hkn)$. Since there are $O(n)$ WL subtrees in level H of \mathcal{R}_s , the complexity of generating \mathcal{E}_{R_n} and \mathcal{E}_{R_g} are $O(kn)$ and $O(n)$, respectively. There are at most $(Hk + 1)n$ vertices in the CRCN relation network. Overall, the time complexity of MRMINE+ is $O(Hm + cn)$, where $c = [H(k + 1) + 1]lr$ (l is the length of truncated random walks, and r is the number of walks sampled per vertex) is a constant much smaller than m, n .

Algorithm 6.4 MRMINE+

Input: Given $\mathcal{G} = \{G_1, G_2, \dots, G_k\}$, the height of subgraph tree in H-CRCN relation network H , the embedding dimension p , the window size w_2 for SkipGram model.

Output: The network embedding matrix \mathbf{F}_g for \mathcal{G} , the subgraph embedding matrix \mathbf{F}_s , and the node embedding matrix \mathbf{F}_n .

```
1: for  $G \in \mathcal{G}$  do
2:   Generate multi-set subtree labels  $\mathcal{L}$  by  $H$  WL iterations.
3: end for
4: Set  $\mathcal{E}_{R_s} = \Phi, \mathcal{E}_{R_g} = \Phi, \mathcal{E}_{R_n} = \Phi$ .
5: for  $(L_G^n)_i \in L_G$  do
6:   if  $(p((L_G^n)_i), p((L_G^n)_i)) \notin \mathcal{E}_{R_s}$  then
7:     Add  $(p((L_G^n)_i), p((L_G^n)_{i+1}))$  to  $\mathcal{E}_{R_s}$ .
8:   end if
9: end for
10: for vertex  $v$  of level  $H$  in  $\mathcal{R}_s$  do
11:   Add  $(v, p_g(G))$  to  $\mathcal{E}_{R_g}$ , if  $q_g(v) \in G$ .
12:   Add  $(v, p_n(n))$  to  $\mathcal{E}_{R_n}$ , if  $q_g(v) \in L^n$ .
13: end for
14: Construct subtree vertices' corpus  $P_s$  with  $\mathcal{E}_{R_s}$ .
15: Construct network vertices' corpus  $P_g$  with  $\mathcal{E}_{R_s} \cup \mathcal{E}_{R_g}$ .
16: Construct node vertices' corpus  $P_n$  with  $\mathcal{E}_{R_s} \cup \mathcal{E}_{R_n}$ .
17: for vertex  $u_i$  in random walk  $r \in P_g \cup P_n \cup P_s$  do
18:    $J(\Phi) = -\log Pr(u_j | \Phi(u_i)), u_j \in r[i - w_2, i + w_2]$ 
19:    $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$ 
20: end for
21: Return embedding matrices  $\mathbf{F}_n, \mathbf{F}_s, \mathbf{F}_g$ .
```

6.1.3 Experimental Results

In this section, we present the experimental results with extensive datasets and baseline methods, to evaluate the effectiveness of handling multi-network mining tasks, and the scalability of the proposed algorithms (MRMINE and MRMINE+).

Experimental Setup. Our proposed method is evaluated mainly on seven real-world datasets, which are summarized in Table 6.2. The brief description of each dataset and the experimental setup are presented as follows.

- *DBLP*: A co-authorship network with nodes representing authors and links representing co-authorship. The original dataset contains 42,252 nodes and 210,320 edges [128].
- *Flickr*: A network of friends on the image and video hosting website *Flickr* with each node representing a user and each edge reflecting friend relationship. It has 215,495 individual users and 9,114,557 friend relationships [129].

Table 6.2: Datasets Summary

Dataset Name	Category	# of Nodes	# of Edges
<i>DBLP</i>	Co-authorship	1,013	3,022
<i>Flickr</i>	User relationship	3,911	4,152
<i>LastFm</i>	User relationship	4,068	4,347
<i>Douban</i>	User relationship	1,118	3,022
<i>MySpace</i>	Social network	6,362	6,514
<i>Aminer</i>	Academic network	1,274,360	4,756,194
Bioinformatics	Size (# of graphs)	Classes	Avg. nodes
<i>MUTAG</i>	188	2	17.9
<i>PTC</i>	344	2	25.5
<i>PROTEINS</i>	1113	2	39.1
<i>NCI1</i>	4110	2	29.8
<i>NCI109</i>	4127	2	29.6

- *LastFm*: Collected in 2013, this is the following network of users on the music website *LastFm*. The network network contains 136,420 users and 1,685,524 following links [129].
- *Douban*: Collected in 2010, this data reflects the users’ friend relationship in the offline and online activities of *Douban*, and contains 50k users and 5M edges. The offline and online activity communities share some overlaps of users, which makes it suitable for network alignment.
- *MySpace*: A social network which has a strong music emphasis. The links between nodes reflect the connections of users. It has 854,498 users and 6,489,736 relationship links.
- *AMiner*: An academic social network. Undirected edges represent co-authorship relationship [129]. The whole dataset contains 1,274,360 nodes and 4,756,194 edges.
- *Bioinformatics*: The bioinformatics dataset, including *MUTAG*, *PTC*, *PROTEINS*, etc. are small-scaled networks of chemical compound, proteins or enzymes, and are often used as benchmark datasets for graph classification.

Using the above datasets, we design the following five experimental scenarios for evaluating the effectiveness of our method.

S1. DBLP vs. Noisy DBLP Alignment. We extract a subnetwork with 1,013 nodes from the original *DBLP* dataset, and randomly add extra edges to the network to generate the second noisy network while keeping the node set unchanged. Note that our setting is different from

that in [30] for that our noisy edges make two networks non-isomorphic while [30] only changes the edge weight.

S2. Douban-offline vs. Douban-online Alignment. We adopt a method introduced in [181] to construct the offline network according to users’ co-occurrence in social gatherings. We treat people as contacts if they participate in the same offline events more than ten times. The constructed offline network has 1,118 users. We use the corresponding online social network for the same 1,118 users as the second network, and add extra nodes as noises in the alignment experiment.

S3. Cross-network Query Node Retrieval. As an complementary experiment for network alignment, we conduct cross-network query node retrieval, in which given a set of query nodes from one network, we aim to retrieve similar nodes from another network. We study the following network pairs and use their overlapping user set as groundtruth. DBLP and noisy DBLP, Douban-offline and Douban-online, Flickr and LastFm, MySpace and noisy MySpace with both node and edge noises.

S4. Collective Network Alignment. To further demonstrate the effectiveness of our method on multi-network mining, we adopt a novel collective network alignment. Rather than conducting traditional two-network alignment, we use three input networks to collectively align nodes of three networks. To the best of our knowledge, we are the first to align more than two networks collectively. We use Douban-offline and Douban-online for this experimental scenario. More details will be elaborated in the next sub-section.

S5. Network Level Classification. We use bioinformatics dataset for graph level classification as shown in [176]. Since our method can learn network, subgraph and node embeddings simultaneously, we can either use network embeddings for classification directly or use node and subgraph embeddings combinatorially as shown in [177]. The results present the performance of the first one since it has better performance out of the two options. After we learn the embeddings, we use 80% of the bioinformatics networks for training and 20% for testing with a linear SVM model.

In all the above scenarios, we do not use network node/edge attributes as auxiliary information in the experiments.

Comparison methods. In total, we use nine comparison methods in our experiments. For (collective) network alignment and query node retrieval experiments, we use three traditional network alignment/matching methods (*FINAL* [30], *IsoRank* [163], and *UniAlign* [81]), and three network embedding methods (*Deepwalk* [173], *node2vec* [172], and *struc2vec* [20]). For network classification experiment, we use three baselines including both traditional Weisfeiler-Lehman kernel method (*WL kernel* [179]) and two embedding-based methods (*Deep Graph Kernel* [177], and *subgraph2vec* [176]).

Repeatability. All of the datasets are public. All experiments are performed on a server with Intel(R) Xeon(R) CPU core with 2.00 GHz and 1.51 TB RAM. The operating system is Red Hat Enterprise Linux Server release 6.9. The algorithms are programmed with Python. The hyperparameters are set based on a grid search. We intend to release the source code after the paper is published.

Effectiveness.

Visualization. To evaluate the effectiveness of the proposed method, we first use a widely used and small-scaled dataset, the Zachary’s Karate Club [20] dataset for visualizing the embeddings in 2-d space for intuitively presenting the difference between our method and baseline embedding methods. We use two identical Karate Club networks in which the second one is permuted from the original network as shown in Figure 6.4. We apply three baseline embedding methods, *Deepwalk*, *node2vec*, and *struc2vec* as well as MRMINE on this dataset. Since all baseline methods only support single network embedding, we fit two networks as one single network into these models. The results are shown in Figure 6.5 and Figure 6.6. As we can see, the embeddings of *Deepwalk* clutter in four positions. Nodes from two networks are mixed together, and can not be distinguished. *node2vec* roughly embeds the nodes into two clusters by nodes’ membership, but the nodes within each cluster are mixed up.

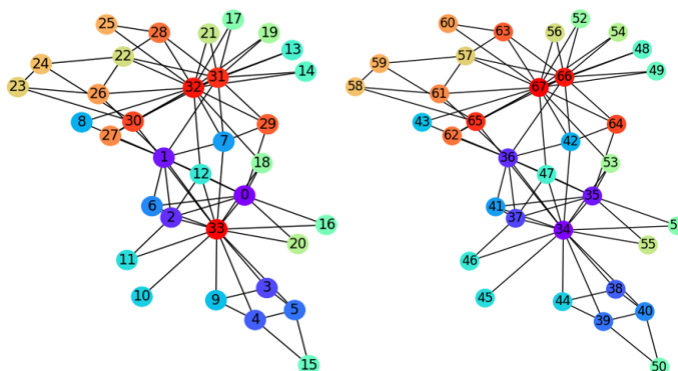


Figure 6.4: Original karate network and permuted karate network. Colors are set the same for identical nodes across two networks.

Also nodes from different networks are incomparable. *struc2vec* embeds nodes based on structural roles, but it can not explicitly differentiate all roles as some nodes with different colors are embedded together, while some nodes with the same color are far apart. Our method pairs almost all nodes with the same colors together, and clusters structurally identical nodes (e.g. node 17, 19, 52, 54 with lime color). This visualization result matches our intuition that our model can preserve structural similarities of nodes across networks and shows great potential in many mining tasks as we will present shortly.

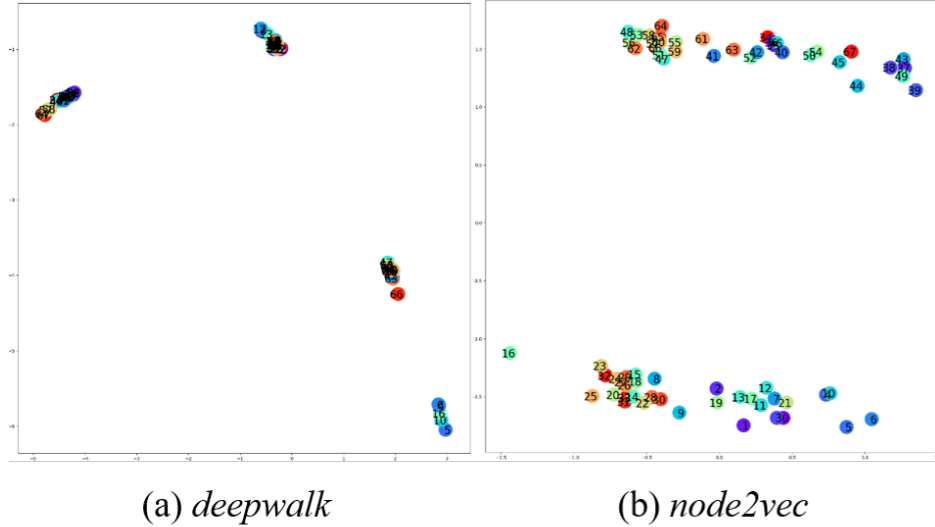


Figure 6.5: 2-d visualization of embeddings learned on Zachary’s Karate Club dataset by two traditional methods *Deepwalk* and *node2vec*. The learned embeddings are 32-d and transformed to 2-d by Multi-Dimensional Scaling (MDS) to preserve the embedding distance. Nodes of same color represent that they are identical. Best viewed in color.

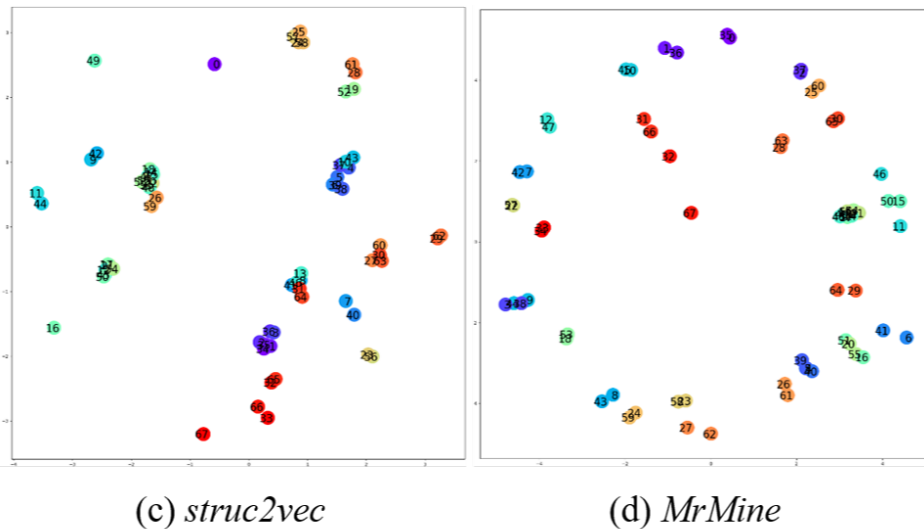
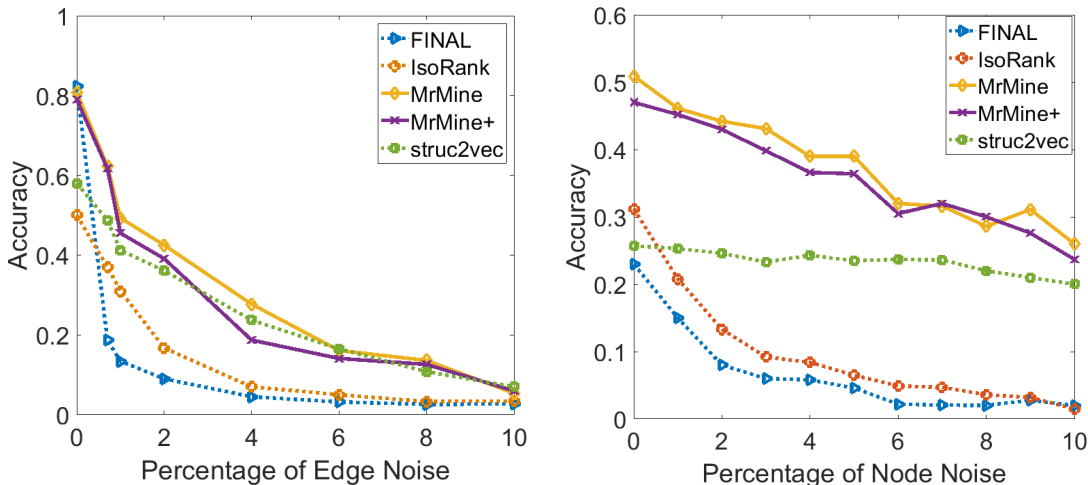


Figure 6.6: 2-d visualization of embeddings learned on Zachary’s Karate Club dataset by traditional method *struc2vec* and by MRMINE. The learned embeddings are 32-d and transformed to 2-d by Multi-Dimensional Scaling (MDS) to preserve the embedding distance. Nodes of same color represent that they are identical. Best viewed in color.

Network Alignment. Next we perform two network alignment experiments (scenario $S1$, $S2$). For traditional network alignment baseline methods (e.g. *FINAL*, *IsoRank*), we calculate the cross-network similarity matrix and apply greedy match algorithm [30] to process the similarity matrix for one-to-one node alignment. For embedding methods, we first generate node embeddings, and calculate the similarity matrix by the inner product of embedding

vectors. The greedy match algorithm is then applied on the similarity matrix. The results are shown in Figure 6.7. As the percentage of edge/node noises increases, the accuracies of all methods decrease.

On both datasets, our proposed methods MRMINE and MRMINE+ outperform all baselines. MRMINE slightly outperforming MRMINE+ indicates the usefulness of cross-network links in the CRCN relation network. Specifically, on *DBLP* dataset, our methods achieve close accuracy to *FINAL* when there is no edge noise, but *FINAL* is sensitive to edge noises and decreases rapidly on small percentage of extra edges. *struc2vec* also achieves close performance to MRMINE+. On *Douban* dataset, our methods outperforms baselines at all node noise level by at most 19.71%.



(a) Alignment result on original *DBLP* and noisy *DBLP* dataset.

(b) Alignment result on *Douban* offline and online network

Figure 6.7: Network alignment result of our methods compared with traditional alignment baseline methods and network embedding baseline methods. Best viewed in color.

Query Node Retrieval. We then conduct query node retrieval experiments (i.e. scenario *S3*). We treat the nodes in one network as queries and the nodes in the other network as targets. For traditional methods (i.e. *FINAL*, *IsoRank* and *UniAlign*), we calculate the cross-network similarity matrix and sort the nodes in the targets based on their similarity values with query nodes. After sorting, top- k nodes are retrieved from targets for each query node. If the matching node exists in the top- k list, we consider it as one hit. The accuracy is calculated as $accuracy = \frac{\text{number of hits}}{\text{number of query nodes}}$. In some recent related works, this evaluation setting is also referred to as the soft network alignment, for it does not output the one-to-one node mapping. We calculate the accuracy of node retrieval w.r.t. the k value and present the results in Figure 6.8. For (a) and (d) in Figure 6.8, we create the dataset by using the original network for the first networks, and inserting 2% edge noises and 3% node noises by

randomly adding edges/nodes to the input networks for the second network. For (b) and (c), we directly use the two partially overlapped different networks as inputs, and use the overlapped node set as queries for node retrieval. We can observe that our proposed method, MRMINE and MRMINE+, significantly outperform all baselines, including both traditional network alignment/matching methods (*FINAL*, *IsoRank* and *UniAlign*) and recent network embedding methods (*Deepwalk*, *struc2vec*) on all four datasets. This experiment reveals the effectiveness of the developed model for multi-network node retrieval in real-world datasets.

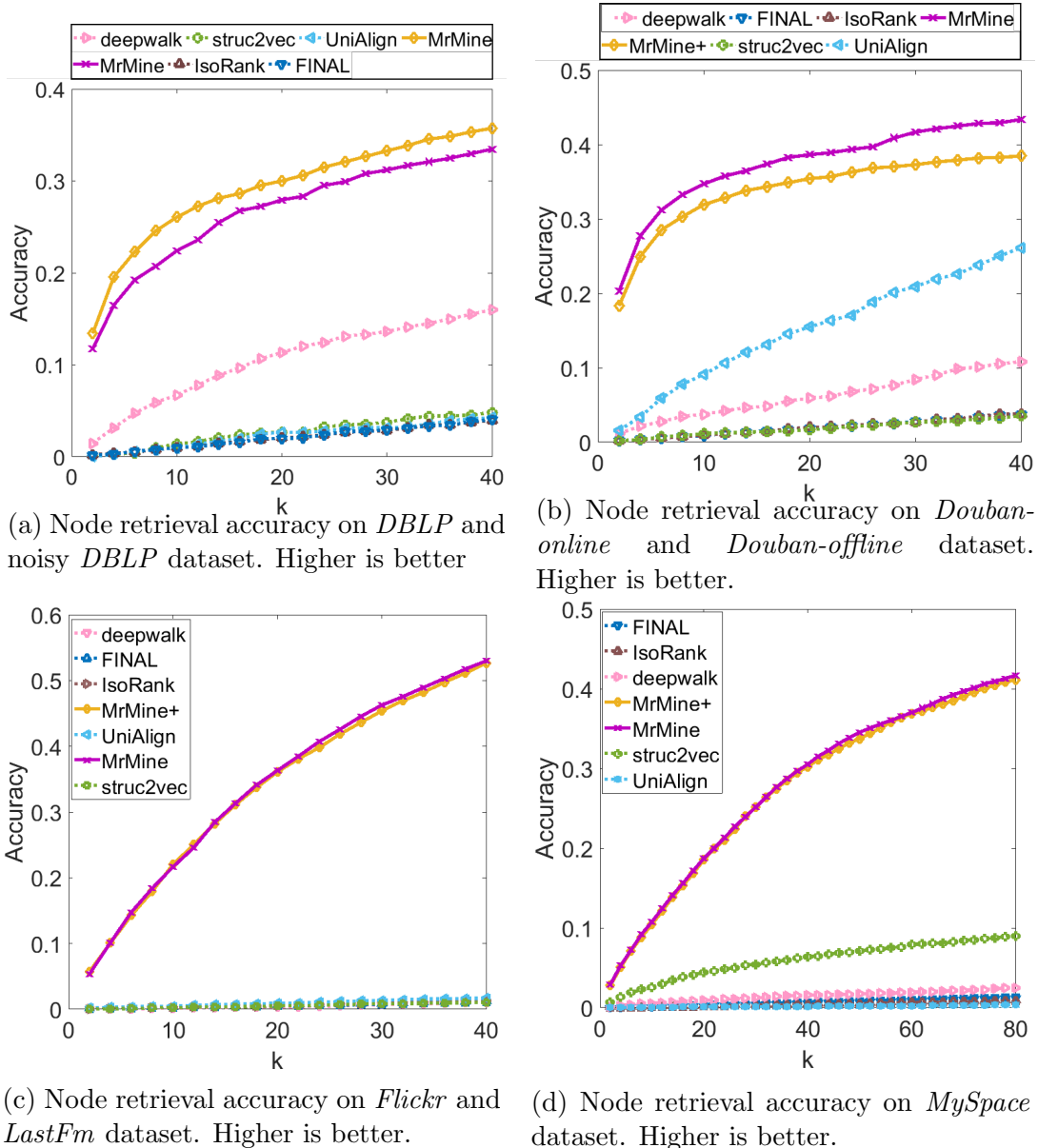
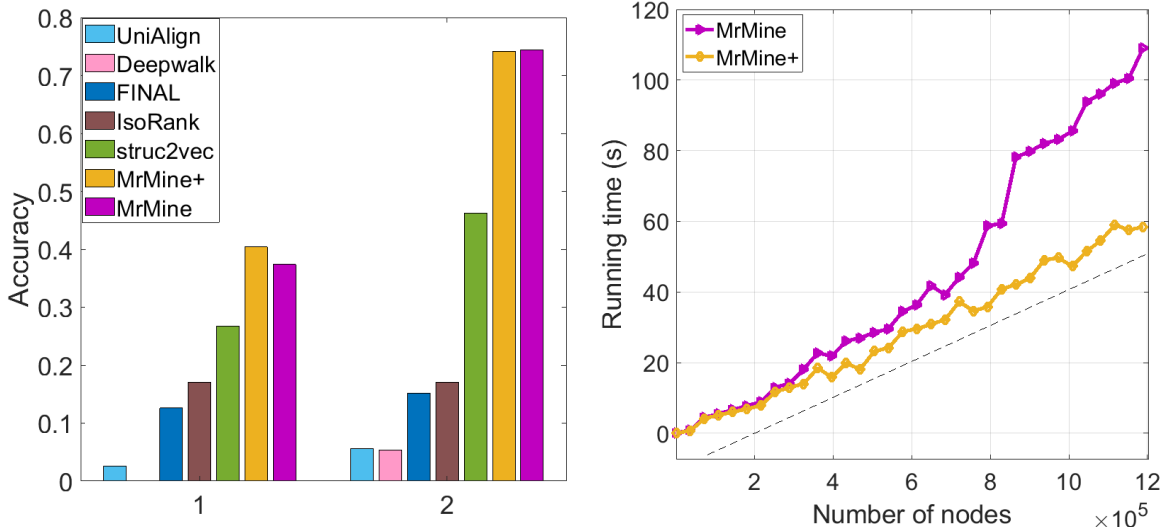


Figure 6.8: Node retrieval results on our methods with five baseline methods. Accuracy vs. top- k retrieved nodes. Best viewed in color.

Collective Network Alignment. We further conduct a novel collective network alignment experiment (scenario S_4) to show the advantage of our methods on enabling difficult multi-network mining task. We collectively perform network alignment among all nodes in three networks (G_1 : *Douban-offline*, G_2 : *Douban-online*, and G_3 : *Douban-online* with 3% edge and 5% node noises). Since traditional network alignment/matching methods only calculate similarity matrix between two networks, we need to calculate \mathbf{S}_{12} for G_1, G_2 , \mathbf{S}_{13} for G_1, G_3 , and \mathbf{S}_{23} for G_2, G_3 . The three-way similarity between G_1, G_2 and G_3 forms a 3-d similarity tensor \mathbf{S} , with $\mathbf{S}(i, j, k) = \mathbf{S}_{12}(i, j) + \mathbf{S}_{13}(i, k) + \mathbf{S}_{23}(j, k)$. We implement a 3-d greedy match algorithm to produce a one-to-one-to-one alignment for nodes in three networks. For our method, since we simultaneously embed all nodes onto the same embedding space, we can directly calculate the similarity tensor \mathbf{S} and apply 3-d greedy match algorithm. We use two metrics for this experiment. First, for each pair of three-node alignment (including three nodes from G_1, G_2 and G_3), we consider it a successful alignment if all nodes are aligned correctly (a strict metric, indicating the correct 3-way alignment). Second, for each pair of three-node alignment, we consider it a successful alignment if two of the three nodes are aligned correctly (a relaxed metric, indicating at least 2-way correct alignment). The results are presented in Figure 6.9 (a). As we can see, our proposed methods MRMINE and MRMINE+ outperforms all baselines by both metrics. The largest accuracy improvement by the strict metric is 14.33%, and the largest accuracy improvement by the relaxed metric is 28.20%.

Network level classification Lastly, we test the performance of the embeddings learned by our proposed methods on network classification. We use all five categories of datasets from bioinformatics dataset, and the results are presented in Table 6.3. The bold numbers show the best performance. Note that different from [176, 177], we do not use node or edge attributes in the embedding learning process. From the results, our methods outperform all baselines in four out of five categories of bioinformatics datasets, and perform very close to the best baseline in the proteins data, which indicates the the proposed methods’ effectiveness of enhancing network level classification tasks. We also observe that MRMINE+ performs consistently better than MRMINE in this classification experiment, which denotes that the cross-network relation captured by H-CRCN relation network is more effective than the basic CRCN relation network in network classification task.

Scalability. We conduct scalability study of the proposed algorithms on the largest dataset *Aminer* which contains over 1M nodes. We use two subgraphs of *Aminer* as input networks, and conduct a series of running time test with the number of nodes ranging from 1,000 to 1.2M (which is close to the size of the entire *Aminer* dataset). Figure 6.9 (b) shows the results of the average running time on 5 runs. We can observe that the running time



(a) 1 shows results by the first (strict) metric and 2 shows results by the second (relaxed) metric. Higher is better. (b) Running time vs. number of nodes (Skip-Gram time is not included) of MRMINE and MRMINE+.

Figure 6.9: Collective alignment results on *Douban* offline and online dataset (a), and scalability study results on MRMINE and MRMINE+ (b). Best viewed in color.

Table 6.3: Comparison of Network classification Accuracy on Bioinformatics Datasets (\pm standard derivation)

	MUTAG	PTC	PROTEINS	NCI1	NCI109
MRMINE+	83.47 \pm 2.01	62.00 \pm 0.07	71.22 \pm 0.62	68.50 \pm 0.03	65.57 \pm 0.02
MRMINE	82.19 \pm 1.58	55.41 \pm 2.52	70.88 \pm 0.38	66.90 \pm 0.05	64.53 \pm 0.01
<i>WL Kernel</i>	80.66 \pm 3.07	59.94 \pm 2.79	64.45 \pm 1.14	63.42 \pm 0.22	62.94 \pm 0.42
<i>Deep WL Kernel</i>	82.95 \pm 1.96	53.29 \pm 1.53	69.49 \pm 0.26	62.83 \pm 0.25	62.47 \pm 0.15
<i>subgraph2vec</i>	79.33 \pm 0.07	42.29 \pm 0.09	73.04 \pm 0.04	63.01 \pm 0.01	49.20 \pm 0.02

of both algorithms are less than 120s when applied on 1.2M-node networks. Particularly MRMINE+ scales linearly while MRMINE scales faster than (super-linearly) MRMINE+ which is consistent with our analysis on the time complexity. Thus, the proposed methods can be applied to large networks.

6.2 HYPERGRAPH REPRESENTATION LEARNING WITH PRE-TRAINING

Hypergraph, as a generalization of the traditional graph data, is ubiquitous in various domains, and has drawn increasing attention recently [14, 77, 182]. Different from traditional graphs, which consist of nodes and edges to represent *pairwise* relations between nodes, each hyperedge contains a collection of nodes, which represents a *high-order* relation. For

example, in the clinical studies of the pharmacological mechanism [182, 183], the effects of medical treatment is often the result of the combined interactions of a set of drugs. Here the combination of drugs for one disease forms one hyperedge. In the bioinformatics research, protein/multi-protein complexes, which consist of different collections of protein molecules, display different functions [184]. Here a collection of protein molecules could be one hyperedge. In the social network domain, a group of users who participate in the same event could be a hyperedge of that event [72, 80]. In the academic domain, authors of the same paper could be a hyperedge of the paper they jointly publish [185].

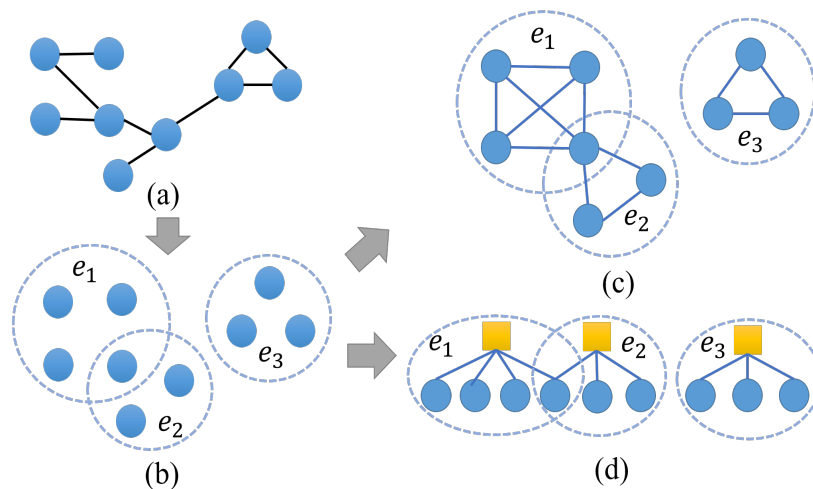


Figure 6.10: (a), (b): An example of plain graph and hypergraph; and (c), (d): two hyperedge expansion methods used in this paper.

Representation learning on hypergraph offers a promising way to streamline various hypergraph applications. However, the traditional graph representation learning methods are not directly applicable in capturing the *high-order* relations of the hypergraphs. To date, relatively few works on hypergraph representation learning exist, most of which focus on hyperlink prediction [77, 80, 100, 182]. The major difficulties of representation learning on hypergraph are two-fold. First, the labels of diverse downstream tasks are usually very scarce, which makes it difficult for training the downstream neural models. Second, most of the recent hypergraph representation learning methods only work in the transductive learning setting [14, 60, 80, 101, 182, 186]. Specifically, these methods require all data to be seen during the training for feature generation or representation learning in the model, which renders the inability of these methods to handle unseen data. For example, in the hyperlink prediction problem, many previous methods require that all candidate hyperlinks to be seen during training.

Beside these limitations, there commonly exists one blind-spot for the dataset. Although

many current hypergraph datasets are constructed from plain graphs, the connections among nodes of the original plain graph should be strictly unavailable in the hypergraph scenario. For example, in Figure 6.10, (a) represents the original plain graph, and (b) represents the hypergraph that is constructed by (a). For such constructed hypergraphs, the black links in (a) should never be known either explicitly (used for model) or implicitly (used for feature generation).

In this section, inspired by the recent advances of pre-training strategies developed in Natural Language Processing (NLP) community [187] and Graph Neural Networks (GNNs) research [188, 189, 190], we propose HyperGRL, a self-supervised pre-training based hypergraph representation learning framework, with the target of both inductive and transductive hyperedge classification. Compared with previous methods, the proposed HyperGRL enjoys the following three distinctive advantages. First, the proposed pre-training framework is capable of leveraging labeled data (with supervised pre-training) as well as unlabeled data (with self-supervised pre-training), to learn transferable knowledge for diverse downstream tasks without the help of extra domain-specific hypergraph datasets [188, 191]. Second, our method explores bi-level (i.e. node-level and hyperedge-level) self-supervised pretext tasks, which aim at capturing the intrinsic *high-order* relationships of nodes and hyperedges respectively. Third, the proposed HyperGRL can work in both transductive and inductive settings. The pre-training strategy proposed for the transductive setting is adaption-aware, in the sense that the pre-trained model could be more adaptive to the downstream tasks compared to traditional pre-training methods, and meanwhile be more computationally efficient.

The main contributions of this section are as follows.

- **Novel Pre-Training Framework.** We propose a bi-level pre-training framework for hypergraph representation learning named HyperGRL, equipped with two mutually complementary self-supervised pretext tasks. The proposed framework can be applied to both transductive and inductive settings. For the transductive setting, the proposed HyperGRL further embraces an adaptation-aware pre-training strategy to accelerate the knowledge transfer.
- **Extensive Empirical Evaluations.** We perform extensive experiments to demonstrate the efficacy of HyperGRL. In particular, the proposed HyperGRL (1) outperforms *all* baselines across all datasets for inductive hyperedge classification, with an up to 5.69% improvement over the best competitor, and (2) improves pre-training efficiency by up to 42.8% on average.

6.2.1 Problem Definition

The main notations used in this section are summarized in Table 6.4. We first define the hypergraphs as follows.

Table 6.4: Symbols and Definition

Symbols	Definition
$G = (\mathcal{V}, \mathcal{E}, \mathbf{F}^{(n)})$	a hypergraph of node set \mathcal{V} , edge set \mathcal{E} , and feature $\mathbf{F}^{(n)}$
\mathbf{M}	the hypergraph incidence matrix
\mathbf{A}	the adjacency matrix of nodes inferred from \mathbf{M}
Θ, Ω_i	parameters of GNN module and adjustment modules
$f_{\Theta}(\cdot)$	GNN module with parameter Θ
$g_{\Omega_i}(\cdot)$	neural adjustment module with parameter Ω_i
$f_{\Theta, \Theta_0 = \Theta'}(\cdot)$	a pre-trained GNN module with initialization Θ'

Definition 6.1. Hypergraph: A hypergraph is represented by $G = (\mathcal{V}, \mathcal{E}, \mathbf{F}^{(n)})$, in which $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ is the set of n nodes and $\mathcal{E} = \{e_1, e_2, \dots, e_m\}$ is a set of m hyperedges. $e_i = \{v_j^{(i)}\}, 1 \leq j \leq n$ represents the i -th hyperedge in which the nodes $v_j^{(i)} \in \mathcal{V}$. We say that node v_j is *inside* hyperedge e_i . $\mathbf{F}^{(n)}$ is the feature matrix¹ for nodes.

A hypergraph incidence matrix [186] $\mathbf{M} \in \mathbb{R}^{n \times m}$ is defined such that $\mathbf{M}(i, j) = 1$ if node i appears in hyperedge j , and $\mathbf{M}(i, j) = 0$ otherwise. From \mathbf{M} , we can build an adjacency matrix $\mathbf{A} = \mathbf{M}^T \mathbf{M}$, in which $\mathbf{A}(i, j)$ indicates the number of nodes that appear in both hyperedge i and hyperedge j .

Before formally defining the inductive hyperedge classification problem, we provide a brief review of Graph Neural Networks (GNNs).

Preliminaries on Graph Neural Networks. GNNs are powerful deep learning models on graphs. Representative models include Graph Convolutional Networks (GCN) [140], Graph Isomorphism Networks (GIN) [142], Graph Attention Networks (GAT) [141], etc. The intuition behind many previous GNN models is to learn the node representation by convolutionally aggregating both the node/edge features and the features of the node’s local neighbors through neural networks. Message passing is often adopted as a popular choice to design various GNN models [192]. There are two main steps in the message passing process, including message passing and message updating. In the message passing step, the node features are passed to its neighbors. In the message updating step, the received features are passed through an aggregation function (e.g., a neural network) for node representations. Typical message passing can be summarized as:

¹Optionally, there might be a feature matrix for hyperedges $\mathbf{F}^{(h)}$.

$$\mathbf{h}_{m_v}^{(t+1)} = P_t(\{\mathbf{h}_v^{(t)}, \mathbf{h}_w^{(t)}, \mathbf{e}_{vw}\}), \forall w \in \mathcal{N}(v) \quad (6.3)$$

$$\mathbf{h}_v^{(t+1)} = U_t(\mathbf{h}_v^{(t)}, \mathbf{h}_{m_v}^{(t+1)}) \quad (6.4)$$

where P_t and U_t are the message passing function and node representation updating function of the t -th iteration respectively. $\mathbf{h}_v, \mathbf{h}_w$ are node representations of neighboring nodes (v, w) , and are initialized as node features. \mathbf{e}_{vw} is the feature of the edge between node v and node w . Different GNN models differ in the functions $P_t()$ and/or $U_t()$. For example, GCN [140] takes the summation of the neighboring nodes in the message passing step and attaches a neural network module on the passed message and the node feature itself for feature aggregation.

By using GNN layer as a neural function $f_{\Theta}(\cdot)$ for node representations, the inductive hyperedge classification problem is defined as follows.

Definition 6.2. Inductive Hyperedge classification: Given a set of hyperedges $\mathcal{E} = \{e_1, e_2, \dots, e_m\}$ which are not seen in the training stage, the goal of GNN model $f_{\Theta}(\cdot)$ is to learn embeddings for the downstream classifier² $g_{\Omega}(\cdot)$ to classify them into t categories. $g_{\Omega}(f_{\Theta}(e_i)) = \mathbf{p}_i, i \in \{1, 2, \dots, m\}$, where \mathbf{p}_i is the prediction vector for e_i with a non-zero entry indicating e_i 's predicted category.

By adopting pre-training strategy, model $f_{\Theta}(\cdot)$ is first trained on pretext task(s). Note that there could be more than one pretext task. The fine-tuning module can be represented as $f_{\Theta: \Theta_0 = \Theta'}(\cdot)$ given the pre-trained GNN module $f_{\Theta'}(\cdot)$. Generally speaking, pre-training $f_{\Theta'}(\cdot)$ could be either supervised if the labels for the pretext task are available, or unsupervised, such as self-supervised methods.

6.2.2 Proposed Pre-Training Framework

Challenges and Key Ideas. The first challenge for pre-training hypergraphs is how to design the self-supervised pretext tasks, since the *high-order* node relations of hypergraphs are significantly different from traditional graphs structurally. Our idea is to incorporate both node-level and hyperedge-level pretext tasks, which aim at capturing both local and global contextual patterns of hypergraphs. Locally, the node inside one specific hyperedge should be distinguished from nodes outside this hyperedge given the context of node. For

² $g_{\Omega}()$ is also known as a neural adjustment module, which is an MLP specified for pretext tasks with parameter Ω for mapping node representations to the predicted labels/values.

one specific hyperedge and a given inside node, we define the context of the node as all other nodes inside the hyperedge as shown in Figure 6.11. Globally, the similarities between hyperedges ought to be preserved. However, calculating pairwise hyperedge similarities itself is challenging and costly, with at least $O(m^2)$ time complexity for calculating every pair of hyperedges if the number of hyperedges is m . As an approximation for learning pair-wise hyperedge similarities, our idea is to first cluster the hyperedges, based on the features of nodes inside hyperedges or the hyperedge adjacency matrix when available, and then to preserve the membership characteristic of the hyperedge clusters.

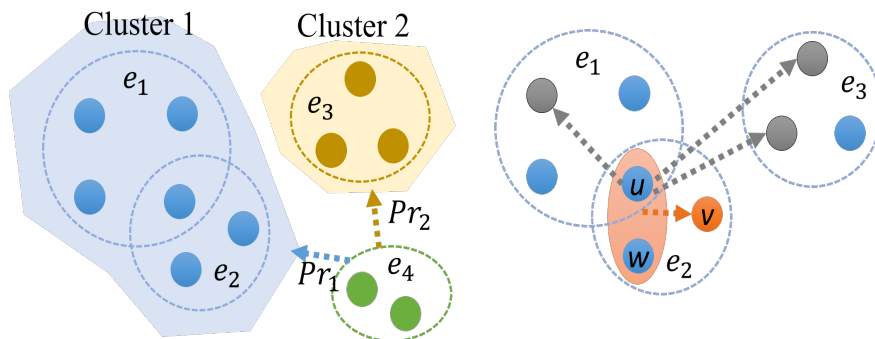


Figure 6.11: An illustrative example of the hyperedge-level pretext task (left), and the node-level pretext task (right). Pr_1, Pr_2 are probabilities for assigning e_4 to cluster 1 and cluster 2. The red area on the right subfigure shows the context of node v , and the gray nodes are the sampled negative examples of node v and its context. Best viewed in color.

The second challenge is how to mitigate the divergence between the self-supervised pretext tasks and the downstream tasks. Even with two mutually complementary pretext tasks, such divergence might still exist, in the sense that a well-trained pre-trained model might be overly fit on the pretext tasks, and could not optimally generalize to the downstream tasks. In the transductive setting, our idea is to design an adaptation-aware pre-training strategy, which targets at learning a well-adaptive pre-trained model for downstream tasks. In this strategy, only one self-supervised pretext task (node-level) is fully trained until convergence, and the other self-supervised pretext task (hyperedge-level) is applied on the unlabeled data for fast adaptation.

Node-level Self-supervised Pretext Task.

Task Description. In this pretext task, we aim at predicting the relationship between a given node and its hyperedge context (i.e., other nodes inside the hyperedge). Intuitively, we expect the node and the context share similar representation if they belong to the same hyperedge.

Specifically, in order to obtain node-level self-supervised training labels, we first uniformly

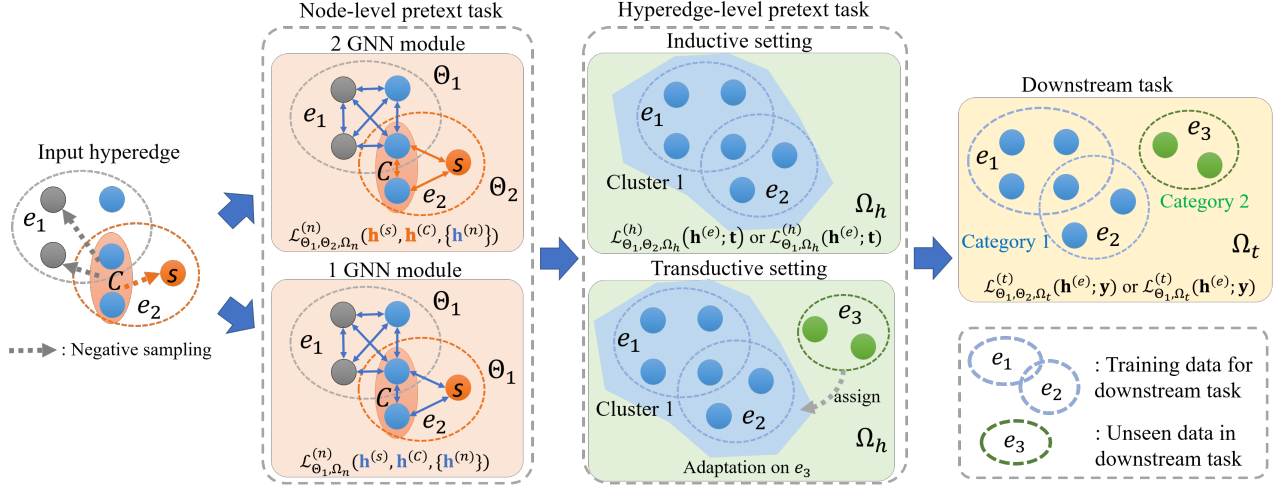


Figure 6.12: The pre-training and fine-tuning framework with node-level and hyperedge-level self-supervised pre-training/adaptation in HyperGRL for hyperedge classification. Best viewed in color.

sample a seed node (e.g. $v_i^{(s)}$) inside each hyperedge, and obtain its corresponding context (e.g., C_i). The combined node-context pair $(v_i^{(s)}, C_i)$ is a positive example. Next, for each pair $(v_i^{(s)}, C_i)$, we adopt a negative sampling method for negative examples ($(v_{ij}^{(n)}, C_{ij}) \sim Pr_{ij}$, the negative sampling probability) of the selected context. We utilize a GNN module inside the *clique-expansion* of hyperedges for the hidden representation of nodes. The representation of nodes corresponding to contexts are aggregated by a pooling layer for the context representations. The node-context relationship is learned via a binary cross-entropy objective. For notation simplicity, let \mathbf{h} be the representation learned by GNN module, and $\hat{\mathbf{h}} := g_{\Omega_n}^{(n)}(\mathbf{h})$ be the node representation after applying the neural adjustment function $g_{\Omega_n}^{(n)}(\cdot)$ of the node-level pretext task.

$$\mathcal{L}^{(n)} = \sum_i \sum_j \log[1 - \sigma((\hat{\mathbf{h}}_i^{(s)})^\top \hat{\mathbf{h}}_i^{(C)})] + \log[\sigma((\hat{\mathbf{h}}_{ij}^{(n)})^\top \hat{\mathbf{h}}_i^{(C)})] \quad (6.5)$$

where $\hat{\mathbf{h}}_i^{(s)}, \hat{\mathbf{h}}_i^{(C)}, \hat{\mathbf{h}}_{ij}^{(n)}$ are the hidden representation of seed node i , context of node i , and the negative sample j of node i after using the adjustment function, respectively. $\sigma(\cdot)$ is a sigmoid function. $(\hat{\mathbf{h}}_i^{(s)})^\top \hat{\mathbf{h}}_i^{(C)}$ is expected to be larger than $(\hat{\mathbf{h}}_{ij}^{(n)})^\top \hat{\mathbf{h}}_i^{(C)}$ since the positive seed node-context pair share similar feature distributions.

Negative Sampling Method. For a given pair of node and its corresponding context in one hyperedge, one naive negative sampling method is to uniformly sample nodes outside this hyperedge. This is applicable even if all hyperedges of a hypergraph do not share nodes, which is often the case in real-world applications. However, this method does not distinguish

between structurally close-by and distant nodes/hyperedges, especially when the relations of nodes from different hyperedges can be obtained via the adjacency matrix \mathbf{A} from the incidence matrix \mathbf{M} . Structurally close nodes tend to have very similar features or even the same labels, and should be avoided as negative samples. We use the following negative sampling strategy such that the structurally close nodes would have exponentially lower probabilities to be sampled for negative nodes selection. Let $\tilde{\mathbf{A}} = \mathbf{A}^k$, where the entries of $\tilde{\mathbf{A}}$ give the number of paths of length k . We normalized $\tilde{\mathbf{A}}$ as $\hat{\mathbf{A}} = \mathbf{D}^{-1}\tilde{\mathbf{A}}$, where \mathbf{D} is a diagonal degree matrix of $\tilde{\mathbf{A}}$, to prevent extremely low probabilities. For a given node i , the probability of sampling node $j \neq i$ is given as follows.

$$Pr_{ij} = \frac{\exp(-\gamma \cdot \hat{\mathbf{A}}_{ij})}{\sum_j \exp(-\gamma \cdot \hat{\mathbf{A}}_{ij})} \quad (6.6)$$

in which $\gamma > 0$ is a scaling scalar for further tuning the sampling probabilities. We consider the nodes which are not reachable by paths of length k having high and equal probability of being sampled. In practice, we find that a small k is often sufficient for good performance, which also helps with the efficiency for calculating and storing \mathbf{A}^k . This negative sampling method is referred to as the exponential sampling method. The positive/negative hyperedges represent the hyperedges from which positive/negative nodes are extracted in the rest of the paper.

Hyperedge-level Self-supervised Pretext Task. In this pretext task, different from the local node-level pretext task, we strive to capture the more global patterns of hypergraphs. As discussed in Section 6.2.2, we aim at predicting hyperedges’ cluster membership information. The clustering is conducted on the graph of hyperedges as follows. First, the graph of hyperedges is constructed with adjacency matrix $\mathbf{A} = \mathbf{M}^T\mathbf{M}$, such that $\mathbf{A}(i, j)$ is the number of nodes that exist in both hyperedges i and j . Next, the METIS algorithm [193] is applied to partition the graph of hyperedges into q clusters. q is empirically selected, and is set to be larger than or equal to the number of categories of hyperedges in the empirical experiments. The GNN module, which shares the parameters with node-level pretext task, is applied inside the *clique expansion* of the hyperedges, and the representations for hyperedges are obtained by a graph pooling layer. Then, by adopting the categorical cross-entropy loss, the hyperedge-level self-supervised task is written as:

$$\mathcal{L}^{(h)} = - \sum_i [\log(\text{softmax}(g_{\Omega_h}^{(h)}(\mathbf{h}_i^{(e)}))) \circ \mathbf{y}_i^{(h)}]^\top \mathbf{1} \quad (6.7)$$

where $\mathbf{h}_i^{(e)}$ is the hidden representation of the hyperedge i , $g_{\Omega_h}^{(h)}(\cdot)$ is a neural adjustment

function to map the hyperedge representations to q -dimensional vector. $\mathbf{y}_i^{(h)}$ is the multi-class label vector of hyperedge i indicating the cluster membership, and $\mathbf{1}$ is an all-one vector, for taking the summation of the $\log(\text{softmax}(\cdot))$ scores of the correct categories.

Adaptation-aware Pre-Training Strategy. Traditional pre-training methods use a two-stage procedure, in which the first stage trains the pretext task until convergence with self-labels, and the second stage trains the downstream task with the task labels. Specifically in our scenario with two self-supervised pretext tasks, the two-stage training can be realized in a serial procedure as follows.

$$\Theta' = \arg \min_{\Theta} \mathcal{L}^{(n)}(g_{\Omega_n}(f_{\Theta}(\mathcal{E}_{train}, \mathbf{F}^{(n)})); \mathbf{y}^{(n)}) \quad (6.8a)$$

$$\Theta^{(pre)} = \arg \min_{\Theta} \mathcal{L}^{(h)}(g_{\Omega_h}(f_{\Theta:\Theta_0=\Theta'}(\mathcal{E}_{train}, \mathbf{F}^{(n)})); \mathbf{y}^{(h)}) \quad (6.8b)$$

$$\hat{\Theta} = \arg \min_{\Theta} \mathcal{L}^{(t)}(g_{\Omega_t}(f_{\Theta:\Theta_0=\Theta^{(pre)}}(\mathcal{E}_{train}, \mathbf{F}^{(n)})); \mathbf{y}^{(t)}) \quad (6.8c)$$

where $\mathcal{L}^{(t)}$ and $\mathbf{y}^{(t)}$ are the loss function and labels for the downstream task respectively. g_{Ω_t} is the neural adjustment module for downstream task. The parameters of GNN module is first obtained by training the node-level pretext task and then by training the hyperedge-level pretext task.

As discussed in Section 6.2.2, the above strategy in the transductive setting might bring non-negligible divergence between pre-training and downstream task training, which would lead to a sub-optimal model. Moreover, since the hyperedge-level pretext task approximately preserves the pair-wise hyperedge distances, pre-training this task till convergence might result in an even larger divergence. To address this issue, we propose an adaptation-aware pre-training strategy. The key idea is only performing node-level pre-training on the hyperedges with downstream task labels, and utilizing the hyperedge-level pretext task as adaptation on the hyperedges whose labels are to be predicted in the transductive setting.

Specifically, we maintain Eq. (6.8a). But instead of training the hyperedge-level pretext task until convergence (Eq. (6.8b)), we use it as an adaptation task, which can be represented as s steps of gradient descent on test hyperedges \mathcal{E}_{test} , and Eq. (6.8b) is replaced by s steps of:

$$\Theta := \Theta - \epsilon \cdot \frac{\partial \mathcal{L}^{(h)}(g_{\Omega_h}(f_{\Theta:\Theta_0=\Theta'}(\mathcal{E}_{test}, \mathbf{F}^{(n)})); \mathbf{y}^{(h)})}{\partial \Theta} \quad (6.9)$$

where ϵ is the learning rate. Compared with the traditional pre-training method, the advantages of the adaptation-aware training strategy are two-fold. First, the pre-trained model for transferring general data knowledge is more adaptive to the downstream tasks. Second, it is a more efficient pre-training strategy, because only one pretext task needs to be fully

trained, and the adaptation, which only requires a few steps, can be conducted whenever the test data³ for the downstream task is available (e.g., in an online system).

Proposed HyperGRL Framework. The end-to-end model architecture is illustrated in Figure 6.12. The negative sampling strategy is first conducted for the input hyperedge set, followed by the node-level pretext task. We propose two variants of GNN modules: (1) the positive and negative hyperedges share the parameters of the same GNN layer; (2) two different GNN layers, which are parameterized as Θ_1 and Θ_2 in Figure 6.12. These variants are applied for the positive and negative hyperedges from which seed node/context representations are generated. This will further distinguish the representations between positive and negative nodes/contexts.

The proposed pre-training framework could flexibly support various types of GNN models, such as GCN [140], GraphSAGE [148], GIN [142], etc. GIN is adopted in the experiments due to its superior empirical performance. After adopting the GNN module, inspired by HGNN [76], which aggregates messages between nodes and hyperedges, we gather messages of nodes for obtaining the context/hyperedge representation. Here, the messages are flowed from nodes to hyperedges, with the aim of: (1) generating hyperedge representations for classification; (2) supporting a more general setting where hyperedges do not share nodes. A pooling layer is used on the aggregated features, such as mean pooling and Set2set [194]. In order to achieve permutation equivariance, set module could be adopted, such as Deep Sets [195]. Next, the hyperedge-level pretext task is adopted in two learning settings. First, in the inductive learning setting, the hyperedge-level pretext task is trained as an additional pre-training stage. If the node-level pretext task uses two GNN modules, the hyperedge representations are the concatenation of the outputs of the pooling layers in two GNN modules. Second, in the transductive learning setting, the hyperedge-level pretext task is used as an adaptation stage (Eq. (6.9)). The fine-tuning for the downstream task follows the pre-training, with the initialization of the pre-trained GNN module.

Complexity Analysis. With the uniform negative sampling method, the major computation of the model is applying GNN module on all hyperedges for training. Taking GIN as an example, the computational complexity is $O(d^2 n' L m \cdot iter)$ where d is the feature dimension, n' is the number of nodes for each hyperedge, L is the number of layers of GNN module, m is the number of hyperedges, and $iter$ is the number of iterations. Here for notation simplicity, we assume all hyperedges share equal number of nodes n' , which is often much smaller than the number of hyperedges m in a hypergraph. For HyperGRL with the exponential negative sampling method, the major computation is calculating $\tilde{\mathbf{A}}^k$ in Eq. (6.6), which takes

³Note that the test data could be observed in the transductive setting, and by self-supervised design, the labels of test data are naturally avoided during training.

$O(kmn')$, where m is the number of non-zero entries in \mathbf{A} . The time complexity of METIS algorithm for hyperedge-level pretext task is $O(m + n' + q \cdot \log(q))$, where q is the number of clusters [193].

Model Variants. We further discuss four variants of the proposed model. We also elaborate two practical implementations of negative sampling, based on the actual set of hyperedges from which the negative sampling method is conducted.

Joint Training. First we briefly describe a natural variant in which the pretext tasks are jointly trained with the downstream task. We also use this training method as a baseline in our experiments. The joint training loss can be written as:

$$\mathcal{L}_{\Theta, \Omega_1, \Omega_2, \Omega_3}^{(joint)} = \alpha \mathcal{L}^{(n)} + \beta \mathcal{L}^{(h)} + \mathcal{L}^{(t)} \quad (6.10)$$

where $\mathcal{L}^{(t)}$ represents the loss function for downstream task, hyperedge classification. α, β are used for re-weighting the bi-level pretext tasks.

Variant I. As discussed in Section 6.2.2, instead of using one GNN module for both positive and negative nodes in the node-level pretext task, one variant is to utilize two GNN modules, which do not share parameters, for positive and negative nodes respectively.

Variant II. For the node-level self-pretext task, we could also learn the hyperedge representation by a ranking-based objective function, which aims at forcing the similar nodes in the feature space to be also close in the representation space, and the dissimilar nodes to having a margin in the representation space. Here we adopt the cosine embedding loss function to replace Eq. (6.5). For node i and context j :

$$\mathcal{L}^{(n)} = \sum_{i,j} (y(1 - \cos(\hat{\mathbf{h}}_i, \hat{\mathbf{h}}_j^{(C)})) + (-y)(\max(0, \cos(\hat{\mathbf{h}}_i, \hat{\mathbf{h}}_j^{(C)}) - \epsilon)) \quad (6.11)$$

where $\epsilon > 0$ is a margin, $y \in \{-1, 1\}$ is the label, and $\cos(\cdot)$ is the cosine similarity function.

Variant III. For the hyperedge-level self-pretext task, in order to approximate the task of preserving the exact pair-wise hyperedge similarities, we can also try to preserve the approximated pair-wise hyperedge distances in a regression objective as follows to replace Eq. (6.7).

$$\mathcal{L}^{(h)} = \|g_{\Omega_2}^{(h)}(\mathbf{H})^\top g_{\Omega_2}^{(h)}(\mathbf{H}) - \mathbf{A}^{k'}\|_F^2 \quad (6.12)$$

where $g_{\Omega_2}^{(h)}(\cdot)$ is a neural adjustment function for the embeddings of hyperedges, \mathbf{H} is the embedding matrix of all hyperedges (with row-wise hyperedge embeddings), $k' > 0$ is a small scalar and $\mathbf{A}^{k'}$ is an approximation for pair-wise hyperedge similarities when the incidence matrix can be used for calculating the adjacency matrix \mathbf{A} . In this Variant, we use the above

pretext task as adaptation steps.

Variant IV. In order to adopt GNN technique and learn the representations for hyperedges. Besides expanding each hyperedge as a fully-connected graph (*clique expansion*), we can also expand a hyperedge as a tree (*tree expansion*). Intuitively, each hyperedge can be seen as a new root node connecting to all the nodes inside the hyperedge. In the message passing of GNN module, the difference between clique expansion and tree expansion is that the nodes in clique expansion first pass their features to the neighbors in the same hyperedge, but the nodes in tree expansion only pass their features to the root nodes. Clique expansion and tree expansion are two typical methods of adding hypothetical connections to the nodes inside hyperedges for expansion.

Connection with Graph Meta-learning. Our proposed pre-training and fine-tuning strategy for hypergraphs is also closely related to the recent Graph Meta-learning methods for Graph Neural Networks. The basic goal for Meta-learning is to first learn a general model on a set of tasks, and then make predictions on target tasks with very few observed examples. For GNN, the idea of Graph Meta-learning is to learn a GNN model on a set of graph mining tasks, by recurrently updating the model parameter via each task [196, 197]. The motivation of Graph Meta-learning is aligned with the motivation of our proposed pre-training strategy. Although in the problem setting of HyperGRL the tasks for learning the general model are unsupervised, the goal is also to learn a model which is generic to be able to adapt towards the downstream tasks. Therefore, the self-supervised pretext tasks could be trained with the downstream tasks in a recurrently fashion [197], which is similar to the training of Graph Meta-learning methods. We note that comparing different training methods and their mechanisms could be a potential future direction.

6.2.3 Experimental Results

In this section, we present the evaluation of the effectiveness and efficiency of the proposed framework on public datasets. The statistics of the datasets are summarized in Table 6.5.

Experimental Setup. Here, we present the experimental results on four public datasets (*Cora*, *Pubmed*, *Corum*, *Disgenet*) to evaluate the proposed model. In particular, we pre-process two versions of *Cora* and *Pubmed* datasets as *Cora/Pubmed-clean/noisy*.

- *Cora*: The Cora dataset consists of 2,708 scientific publications (nodes) classified into one of seven classes. The citation network consists of 5,429 edges between publications. The dataset contains text features for each publication, and it is described by a zero/one-valued word vector indicating the absence/presence of the corresponding

Table 6.5: The summary of datasets

Name	# of nodes	# of hyper-edges	Min # of nodes in hyperedge	Max # of nodes in hyperedge
<i>Cora</i>	2,708	2,427	2	169
<i>Pubmed</i>	19,717	3,887	3	20
<i>Corum</i>	6,132	4,736	2	131
<i>Disgenet</i>	8,352	8,386	3	487

word from the dictionary. The dictionary consists of 1,433 unique words [185]. Note that the *Cora* dataset is a traditional graph dataset.

- *Pubmed*: The Pubmed Diabetes dataset consists of 19,717 scientific publications (nodes) from PubMed database pertaining to diabetes classified into one of three classes. The citation network consists of 44,338 links. This dataset contains text features for each publication which is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words [198]. The *Pubmed* dataset is also a traditional graph dataset.
- *Corum*⁴: The Corum is the dataset of mammalian protein complexes. The dataset contains 6,132 types of proteins (nodes), and each protein complex consists of a collection of proteins. No direct connections between proteins in the complexes exist. The *Corum* dataset is a hypergraph dataset.
- *Disgenet*⁵: The dataset contains 8,352 genes (nodes) and each disease (hyperedge) consist of a collection of genes. Each disease is classified into one of 23 MeSH codes. 21 of these codes are used as the hyperedge categories. Note that this dataset is highly unbalanced.

Dataset processing. Since not all of the above datasets are originally hypergraph datasets (i.e. *Cora* and *Pubmed*), we need to first process them for generating the hypergraphs. For *Cora* and *Pubmed*, we generate two versions of hypergraph datasets as follows. For the first version, we take the ego-network (subgraph of the center node with 1-hop neighbors) of each node as hyperedges, and assign the hyperedge label as the label of the majority in the ego-network. We name this version as the noisy version since the hyperedge might contain

⁴<https://mips.helmholtz-muenchen.de/corum/>

⁵<https://www.disgenet.org/downloads>

nodes with different categories. For the second version, we also take the ego-network of each node as hyperedges, but only keep those whose nodes share the same category. We name this version as the clean version. For *Corum* and *Disgenet*, they are originally hypergraph datasets, and do not need processing.

As for the node features, for *Cora* and *Pubmed*, we use the text feature vectors as described in the dataset details. For *Disgenet*, we use the numerical features of genes from the original data (e.g. DSI, DPI, etc.) For *Corum*, we first construct a traditional graph of nodes, in which each edge represents that the end nodes exist in the same hyperedge. Then we adopt the Subsample and Traverse (SaT) Walks [101] strategy for sampling a collection of random walks and use the embedding vectors from *Deepwalk* [173] method as node features⁶.

Baselines and adjustment. In total we adopt six baselines in our experiments. Five of them are from previous work (*Deep Hyperedge* [101], *DHNE* [100], *Hyper-SAGNN* [77], *Graph-SAGE* [148], *Deepwalk* [173]), and one of them is the joint training strategy with the proposed self-supervised pretext tasks (Eq. (6.10)). Among the baselines from previous work, only *Deep Hyperedge* is directly designed for hyperedge classification. *DHNE* and *Hyper-SAGNN* are originally designed for hyperlink prediction. We keep the major model architecture, and adapt these two methods as follows.

For *DHNE*, firstly the second-order component is designed for heterogeneous hyperedges. Since our datasets are not heterogeneous, we only need to use one auto-encoder in this component. Second, the supervised binary component for hyperlink prediction is modified to multi-class hyperedge classification. For *Hyper-SAGNN*, we keep the idea of using both static and dynamic embeddings of nodes, and take the summation of the static and dynamic embeddings as the final embeddings of hyperedges for hyperedge classification.

Graph-SAGE and *Deepwalk* are originally designed for traditional graphs. We adapt these two models to make them work on the traditional graph of hyperedges. In this traditional graph, each vertex represents one hyperedge and there is an edge between two vertexes if two hyperedges share nodes.

Reproducibility.

Model Training Details. First, for HyperGRL with uniform negative sampling, we adopt a local sampling method for the negative hyperedges in order to sample negative nodes. Specifically, the uniform sampling is conducted inside each batch of hyperedges of the stochastic gradient descent algorithm (e.g. Adam optimizer). To this end, the node-level self-supervised pretext task captures the local node-context relationship within a batch at each parameter updating. Since the batch is uniformly sampled, this local sampling method eventually

⁶Note that for all baselines (except for *Deepwalk* which does not utilize node features), we use the same node features as model inputs for fair comparison.

equals to the global uniform sampling on all hyperedges for pre-training. Compared with exponential sampling, which calculates $\tilde{\mathbf{A}}^k$ (Eq. (6.6)) for globally sampling negative nodes, this uniform negative sampling is more efficient (see Figure 6.15).

Second, in order to obtain the hyperedge embeddings from the node embeddings as the outputs of GNN module, mean pooling is adopted on public datasets and Set2set [194] pooling with 1 recurrent layer is adopted on case study because of optimal practical performance. Other sophisticated pooling methods for hypergraphs could be one future direction.

Third, all the adaptation functions which take the node/hyperedge embeddings as inputs for adapting with the self-supervised pretext/downstream tasks are set as MLPs with 0.5 dropout rate. All non-linear activation functions are set as ReLU function.

Hyper-parameter Setting. For the model optimization, we adopt Adaptive Moment Estimation (Adam). The hyper-parameters are set such that the downstream task perform well on validation set. Specifically, for the optimizer, we select the batch size as 64, and learning rate as 0.001 with learning rate scheduler that reduces learning rate on plateau. The number of epochs for node-level pretext task and hyperedge-level pretext task (for traditional pre-training) are set to 50. For the effectiveness evaluations on public datasets, the number of clusters for hyperedge-level pretext task is set as 10, 10, 3, 3, 21, 20 for *Cora-noisy*, *Cora-clean*, *Pubmed-noisy*, *Pubmed-clean*, *Corum*, *Disgenet* datasets, respectively. The number of GNN layers is set as 1. The number of adaptation steps are set to 5 for all datasets. The dimension of node/hyperedge embeddings are set as 64.

For baseline methods, the hyper-parameters are empirically optimized based on the literature. For joint training, we set the weighting parameters α, β to be equal to 1 in Eq. (6.10). For *Hyper-SAGNN*, *DHNE*, *SAGE*, and *Joint Training* methods, we train them using Adam optimization method as well. The learning rate are set as 0.001 with learning rate scheduler. The dimension of node/hyperedge embeddings is set as 64. Other baseline hyper-parameters are set either based on the validation set or original literature guidance.

Effectiveness Results on Public Datasets. The results for hyperedge classification are presented in Table 6.6. The metric is the multi-class classification accuracy, and the results are mean and standard deviation values over ten runs. All supervised methods share the same training, validation, and testing ratio of 6:2:2. The best results are shown in bold fonts, and the second best results are shown with underlines. We adopt the two-GNN module for the inductive setting (no Ada. in Table 6.6), and one-GNN module for the transductive setting (Ada. in Table 6.6). The GNN module here is GIN. Both exponential and uniform sampling methods are evaluated (shorted as Exp. and Uni. in Table 6.6). For baselines, *Hyper-SAGNN*, *SAGE*, *DHNE* and *Joint Training* use the inductive setting. *Joint Training* is a proposed baseline which trains the two pretext tasks together with the downstream task

Table 6.6: Accuracy of hyperedge classification (mean \pm std in %). Bold and underline values indicate the best and the 2-nd best performance respectively. We also conduct a significance test. For all the tables, \bullet / $*$ indicates the result is significantly better/worse than the 2-nd best/the best model with p-value < 0.01 , and \circ indicates no significant difference.

Models	Cora-noisy	Cora-clean	Pubmed-noisy	Pubmed-clean	Corum	Disgenet
DHNE	69.85 \pm 1.01	72.48 \pm 0.52	78.65 \pm 1.59	83.23 \pm 0.74	53.11 \pm 1.39	30.35 \pm 0.83
Hyper-SAGNN	66.94 \pm 2.12	67.81 \pm 1.25	83.20 \pm 2.00	83.82 \pm 0.62	79.64 \pm 0.29	31.74 \pm 0.06
SAGE	72.51 \pm 1.78	76.01 \pm 1.48	83.37 \pm 2.32	78.84 \pm 1.88	56.22 \pm 2.43	18.31 \pm 1.37
Joint Training	70.84 \pm 0.67	81.65 \pm 1.16	85.39 \pm 1.48	86.45 \pm 0.98	76.85 \pm 0.77	34.02 \pm 1.28
DW	73.39 \pm 1.64	81.66 \pm 1.41	82.73 \pm 1.70	<u>88.54\pm0.62</u>	67.35 \pm 2.18	29.07 \pm 1.45
Deep-Hyperedge	74.35 \pm 0.64	72.66 \pm 0.93	64.33 \pm 0.71	81.58 \pm 1.01	82.74 \pm 1.64	35.46\pm1.08
HyperGRL (Uni., no Ada.)	<u>77.98\pm1.81</u> \bullet	86.41\pm1.62 \bullet	85.18 \pm 0.68 \circ	88.21 \pm 1.51 \circ	84.07 \pm 0.70 \bullet	33.23 \pm 1.45 $*$
HyperGRL (Exp., no Ada.)	77.68 \pm 2.67 \bullet	81.74 \pm 1.14 \circ	<u>86.29\pm1.50</u> \bullet	87.81 \pm 0.49 $*$	<u>84.49\pm0.47</u> \bullet	34.10 \pm 0.83 $*$
HyperGRL (Uni.)	74.53 \pm 1.31 \bullet	<u>83.86\pm3.55</u> \bullet	85.52 \pm 0.95 \circ	87.23 \pm 0.96 $*$	84.31 \pm 0.99 \bullet	<u>35.05\pm0.49</u> \circ
HyperGRL (Exp.)	78.78\pm1.28 \bullet	83.77 \pm 1.62 \bullet	86.94\pm0.73 \bullet	90.84\pm0.29 \bullet	85.33\pm1.09 \bullet	33.90 \pm 2.26 $*$

in a joint loss. *DW* and *Deep-Hyperedge* use the transductive setting.

From the table, we make the following observations. First, for the inductive setting, HyperGRL significantly outperforms all baselines on all datasets (by up to 5.69%), and for the transductive setting, HyperGRL significantly outperforms all baselines on five out of six datasets by up to 4.75%. The framework with adaptation-aware pre-training outperforms the traditional pre-training method in five out of six datasets. The exponential sampling method is competitive with the uniform negative sampling method when no adaptation stage is used, but it shows improvements when adaptation-aware pre-training is applied. The performance of the joint training model is competitive compared with the baselines. Among the baselines, *Deep Hyperedge*, *Hyper-SAGNN* and *Deepwalk* have relatively better performance over the rest of the baseline methods, because *DHNE* is originally designed for hyperlink prediction. Note that *Deep Hyperedge* requires hyperedge association as input. HyperGRL does not use such information in downstream tasks, but still outperforms *Deep Hyperedge* in five datasets, and is also competitive on *Disgenet*. For *Disgenet*, the relatively low accuracy is mainly due to the highly imbalanced data with 21 hyperedge categories.

Ablation Study. First, we compare the hyperedge classification performance by using different GNN modules in our framework. For all the different versions of GNN modules,

Table 6.7: Ablation study on bi-level self-supervised pretext tasks (mean \pm std in %)

Models	Cora-noisy	Cora-clean	Pubmed-noisy	Pubmed-clean	Corum	Disgenet
No Pre-train	69.12 \pm 0.48	81.48 \pm 1.24	84.40 \pm 1.53	84.60 \pm 2.49	78.25 \pm 0.13	32.76 \pm 1.54
Only Node	74.23 \pm 0.84	82.27 \pm 1.25	85.04 \pm 0.37	86.84 \pm 1.94	82.52 \pm 2.14	33.75 \pm 1.70
Only Hyperedge	73.24 \pm 1.53	82.78 \pm 1.30	86.29 \pm 1.36	85.67 \pm 0.38	84.17 \pm 1.56	33.40 \pm 0.65
HyperGRL (Exp.)	78.78\pm1.28 •	83.77\pm1.62 •	86.94\pm0.73 ◦	90.84\pm0.29 •	85.33\pm1.09 •	33.90\pm2.26 ◦

Table 6.8: Accuracy of hyperedge classification for different GNN module (mean \pm std in %)

Models	Cora-noisy	Cora-clean	Pubmed-noisy	Pubmed-clean	Corum	Disgenet
GCN (Uni.)	75.58 \pm 0.98	82.09 \pm 1.26	83.59 \pm 0.63	85.96 \pm 2.27	65.23 \pm 0.48	32.36 \pm 0.44
GCN (Exp.)	<u>76.88\pm2.08</u>	84.65\pm3.32	84.14 \pm 1.62	86.06 \pm 1.94	83.34 \pm 0.25	32.84 \pm 0.18
SAGE (Uni.)	75.28 \pm 0.69	80.68 \pm 0.94	83.89 \pm 1.27	86.15 \pm 2.71	64.98 \pm 0.45	33.33 \pm 0.79
SAGE (Exp.)	75.52 \pm 1.01	81.04 \pm 2.09	85.18 \pm 0.86	86.94 \pm 0.32	66.03 \pm 0.28	31.69 \pm 0.21
GAT (Uni.)	73.12 \pm 2.56	81.31 \pm 0.24	84.15 \pm 1.49	86.74 \pm 2.67	82.70 \pm 2.47	31.89 \pm 2.38
GAT (Exp.)	71.21 \pm 0.72	80.15 \pm 1.68	84.79 \pm 0.72	<u>88.49\pm0.72</u>	85.75\pm1.29	29.89 \pm 1.48
GIN (Uni.)	74.53 \pm 1.81	<u>83.86\pm3.55</u>	<u>85.52\pm0.95</u>	87.23 \pm 0.96	84.31 \pm 0.99	35.05\pm0.49
GIN (Exp.)	78.78\pm1.28 •	* 83.77 \pm 1.62 *	• 86.94\pm0.73 •	* 90.84\pm0.29 •	* <u>85.33\pm1.09</u> *	• <u>33.90\pm2.26</u> •

we apply the adaptation-aware pre-training strategy, and the exact same hyper-parameters for the rest of the framework. The results are shown in Table 6.8. The results are the mean and standard deviation values of ten runs. We can see that the GIN module overall shows the best performance over the rest of the GNN modules in our framework for the hyperedge classification task. Also, for the same GNN module, we can see that generally the exponential negative sampling method outperforms the uniform negative sampling method.

Second, we conduct the ablation study on the transductive hyperedge classification performance for bi-level self-supervised pretext tasks (Table 6.7). We apply HyperGRL without any pre-training stage (the first row), with only node-level self-supervised pretext task (the second row), and with only hyperedge-level self-supervised pretext task (the third row). We can observe that the HyperGRL framework with bi-level self-supervised pretext tasks shows the best performance, which demonstrates the effectiveness of both levels of self-supervised pretext tasks.

Parameter Sensitivity Results. Here, we study the hyper-parameter sensitivity of our proposed framework. First, we show the results of the sensitivity of the number of clusters

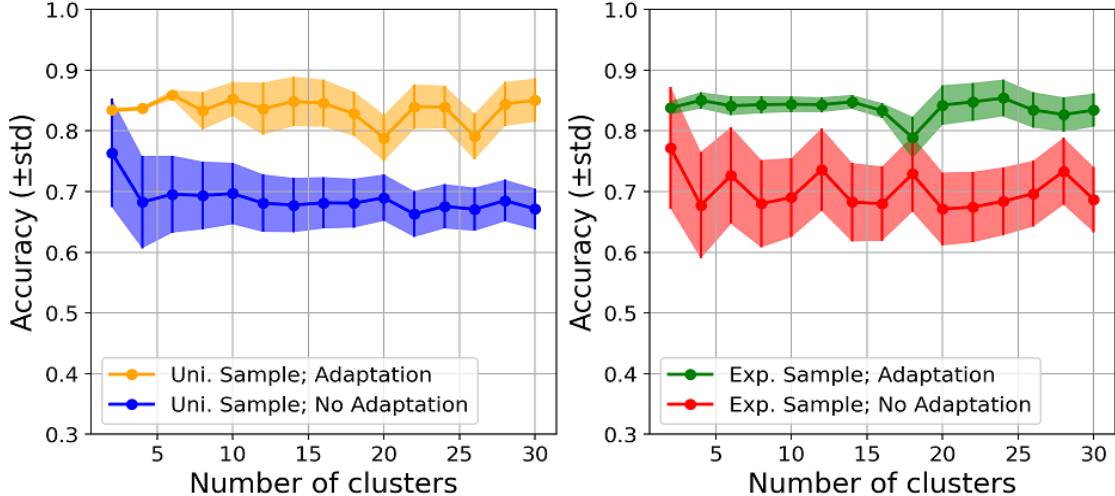


Figure 6.13: Accuracy (\pm std) vs. # of clusters in Hyperedge-level pretext task on *Corum* dataset.

in the hyperedge-level pretext task in Figure 6.13. Note that we use the same model architecture with one-GNN module for all experiments in this subsection. The experiments are conducted in both inductive and transductive settings, with both uniform and exponential negative sampling methods. We observe that: (1) the framework shows relatively stable performance on a large range of the number of clusters; and (2) the proposed framework that uses the adaptation-aware pre-training strategy overall shows more stable performance compared with the framework that does not use the adaptation-aware pre-training. Exponential sampling with adaptation-aware training generally has the best stability. This indicates that the bi-level adaptation-aware pre-training framework has a better adaptation ability for the downstream task. Second, the results of hyperedge classification accuracy vs. the number of adaptation steps are shown in Figure 6.14. The results demonstrate slight performance drop but relatively stable over the tested adaptation steps in the range of $[1, 20]$ for both exponential and uniform negative sampling methods.

Efficiency Results. We compare the efficiency of the adaptation-aware pre-training strategy with the traditional pre-training method as we describe in Subsection 6.2.2. The running time comparison of the pre-training stage is presented in Figure 6.15. For both methods, the number of node-level pre-training is set equal until convergence. The number of adaptation steps for adaptation-aware pre-training method is set to 5, which is the same as the setting in effectiveness evaluations. As we can see, adaptation-aware pre-training significantly reduces the pre-training time by 38.2% with Exp. sampling and 42.8% with Uni. sampling. Besides, as we adopt local sampling in the uniform negative sampling method, the running time for uniform negative sampling achieves ~ 7 times reduction compared with exponential negative

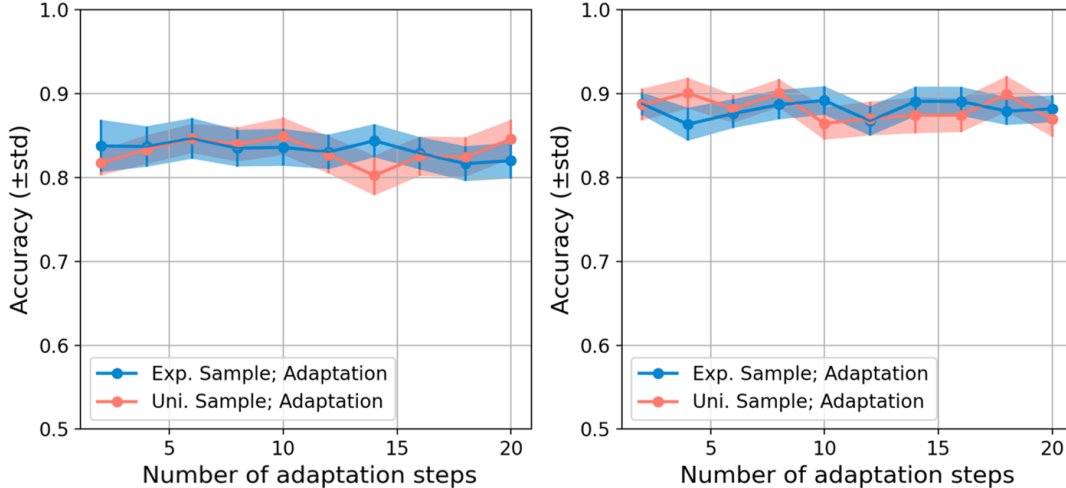


Figure 6.14: Accuracy (\pm std) vs. # of adaptation steps on *Cora-clean* (left) and *Pubmed-clean* (right) dataset.

sampling.

Additional Experimental Results. We present the experimental results on the four variants which we discuss in Table. 6.9. Compared with HyperGRL with exponential negative sampling, the Variant-I with two GNN modules in transductive setting achieves slightly better performance in *Cora* dataset. Variant-II and Variant-III with different node-level/hyperedge-level pretext task objectives could not improve the performance of HyperGRL. Variant-IV with tree expansion could also not achieve improvements over original methods. Further sophisticated model architectures and variants are left for future work. Figure 6.16 shows the parameter sensitivity of the number of clusters on *Cora-clean* dataset. Similar to the observations from Figure 6.13, Figure 6.16 shows that the framework has relatively stable performance on a large range of the number of clusters; and using exponential negative sampling method with adaptation-aware pre-training strategy overall has stabler performance compared with using uniform negative sampling without the adaptation-aware pre-training.

6.3 A CASE STUDY: INCONSISTENT VARIATION FAMILY DETECTION

In this section, we introduce a real-world application of the proposed model on the inconsistent variation family detection problem in a large online store, and show that how the proposed pre-training strategy can help improve a base classification model (without pre-training) as well as other baselines which do not use pre-training, even when direct hyperedge associations are not available (i.e. hyperedges do not share nodes).

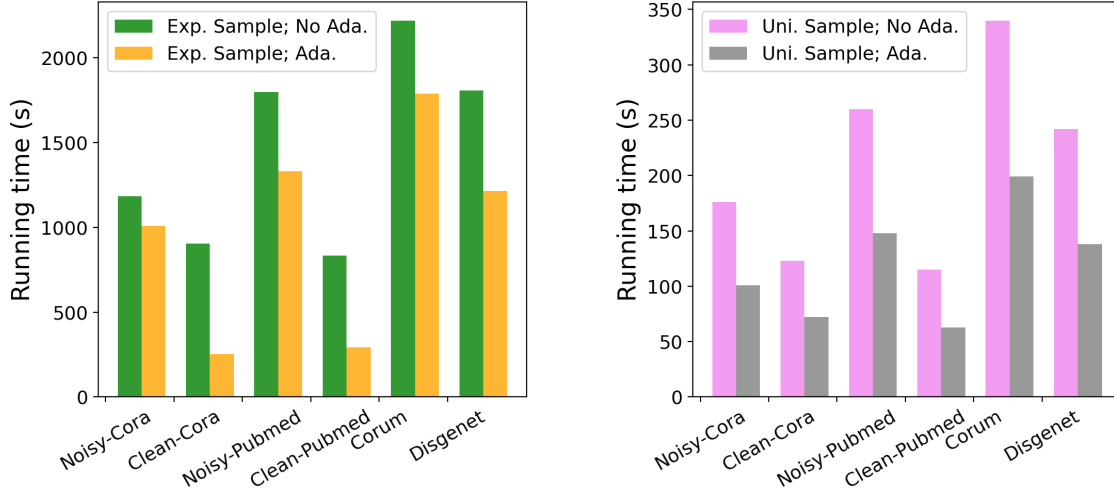


Figure 6.15: Running time comparison of the pre-training stage for traditional and adaptation-aware pre-training strategy.

Table 6.9: Comparison of transductive hyperedge classification performance for model variants

Models	Cora-noisy	Cora-clean	Pubmed-noisy	Pubmed-clean	Corum	Disgenet
Variant-I	78.97±3.69	84.13±2.65	84.72±1.22	88.20±0.68	84.65±0.05	33.00±2.69
Variant-II	76.20±2.21	81.84±0.65	83.25±0.66	87.42±1.24	82.98±1.81	30.72±0.76
Variant-III	74.17±1.82	82.19±2.16	85.09±0.92	88.11±0.90	83.12±0.31	31.54±2.39
Variant-IV	76.56±1.43	82.64±0.39	85.56±0.39	87.32±0.97	68.24±1.41	32.15±2.10
HyperGRL (Exp.)	78.78±1.28 ○	83.77±1.62 *	86.94±0.73 ●	90.84±0.29 ●	85.33±1.09 ●	33.90±2.26 ●

6.3.1 Preliminaries and Experimental Setup

Preliminaries. Large E-commerce services such as Amazon, Etsy, and eBay often utilize merchandise relationships such as variations and substitutes to improve their catalog quality. These relationships between merchandise items are the cornerstone in the field of relationship science.

Variation family refers to a family of product items which are functionally the same but differ in specific attributes. Such variation families are present in the same detail page. For example, in Figure 6.17 the correctly grouped ‘Nike Air Max 270 React’ shoe family is shown in one detail page. All items inside this family are this particular model but have different sizes and colors. The grouping attributes of a variation family are shared by all items inside the variation family (e.g., brand ‘Nike’), and the variation theme attributes are attributes which differ from item to item inside the variation family (e.g., size and color).

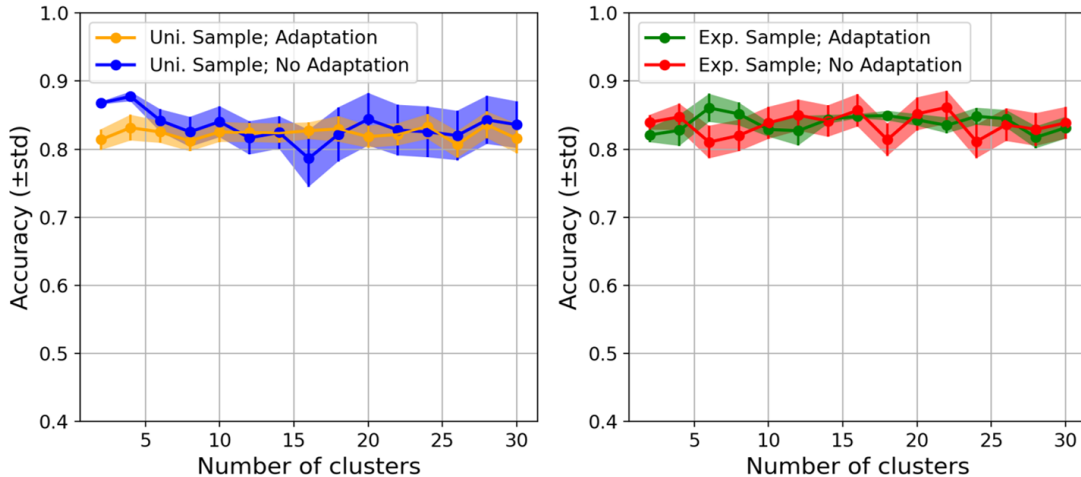


Figure 6.16: Accuracy (\pm std) vs. # of clusters in Hyperedge-level pretext task on *Cora-clean* dataset

When variation families contain inconsistent items (e.g., an Adidas shoe is grouped into the variation family in Figure 6.17), they are called inconsistent variation family (IVF). One of the most important tasks in the catalog system is the IVF detection.

In order to solve this problem, each variation family can be regarded as a hyperedge, in which every item should belong to the same merchandise if it is a consistent family. Then, this specific problem is transformed into hyperedge classification problem. As for the pretext tasks, we adopt a task called variation theme learning, which is defined as:

Definition 6.3. Variation Theme Learning: Given a set of variation families (hypergraph $G = (\mathcal{V}, \mathcal{E}, \mathbf{F}^{(n)})$ with $\mathcal{E} = \{e_1, \dots, e_m\}$), the Variation Theme Learning aims at learning the variation theme attributes and grouping attributes of the families (i.e. learning which columns of $\mathbf{F}^{(n)}$ correspond to grouping/variation theme attributes).

Note that variation families are independent, which makes the direct hyperedge associations unavailable. Most baselines from Section 6.2.3 become inapplicable, such as *SAGE*, *DW*, *Deep-Hyperedge*, and *DHNE*. Our idea is to transform this problem as a hyperedge multi-label classification problem since variation families (hyperedges) might have multiple

Table 6.10: Statistics of Datasets for Case Study

Name	# of families	# of items	# of unique items
Category 1	1,186	64,752	9,540
Category 2	2,169	48,860	8,474
Category 3	4,247	138,662	21,131

Table 6.11: Results of IVF classification for case study (mean \pm std in %)

Dataset	Category 1		Category 2		Category 3	
Metric	AUC-I	AUC-C	AUC-I	AUC-C	AUC-I	AUC-C
Hyper-SAGNN	56.12 \pm 1.19	88.87 \pm 0.93	58.72 \pm 1.94	89.81 \pm 1.64	70.26 \pm 1.12	88.47 \pm 0.78
Hyper-SAGNN, pre-train	60.26 \pm 1.42	90.12 \pm 0.38	<u>59.82\pm0.55</u>	89.57 \pm 0.21	72.17 \pm 0.34	90.17 \pm 1.42
HyperGRL, no pre-train	60.51 \pm 0.89	90.32 \pm 0.88	58.09 \pm 0.57	89.64 \pm 0.79	71.50 \pm 1.49	89.80 \pm 1.72
Joint Training	<u>62.80 \pm 1.33</u>	90.99\pm0.78	56.73 \pm 1.41	89.55 \pm 0.82	71.50 \pm 1.52	89.80 \pm 1.39
HyperGRL, pre-train	64.06\pm1.27 ●	<u>90.74\pm1.40</u> ○	59.42 \pm 1.78 ○	<u>89.84\pm0.70</u> ○	<u>74.32\pm0.81</u> ●	90.62\pm1.53 ●
HyperGRL, pre-train + self-ada.	61.86 \pm 1.32 ○	90.31 \pm 1.80 ○	61.73\pm1.24 ●	90.42\pm1.74 ●	75.42\pm2.01 ●	<u>90.32\pm1.85</u> ○

grouping/variation theme attributes. Due to the fact that the labels of grouping/variation theme attributes are available in our catalog system for variation family datasets, this pretext task in the pre-training stage could be supervised. During training, the hyperedge representation produced by HyperGRL is first used for the multi-label supervised pretext task (variation theme learning), and then the model is fine-tuned by the IVF detection task⁷. IVF detection task has binary labels indicating consistent and inconsistent families (binary classification).

Experimental Setup. We sample subsets of three product lines from the catalog system, named Category 1, Category 2, and Category 3. Some details of these datasets are show in Table 6.10.

The proposed method we select for this experiments is HyperGRL with supervised pre-training task, and HyperGRL with supervised pre-training task plus self-supervised node-level adaptation. Here the supervised pre-training task denotes variation theme learning. For comparison, we use three strong baselines, including the joint training method, which jointly trains the pretext task (variation theme learning task) with the targeted IVF classification (denoted as ‘Joint training’ in Table 6.11), HyperGRL model without pre-training stage, and the *Hyper-SAGNN* model with and without the help of our proposed pre-training strategy. Note that the original *Hyper-SAGNN* is not a pre-training model. We modify the model to adapt it in our pre-training framework as a strong baseline. Besides the above adaptation,

⁷As suggested by [188], these two tasks are highly likely to be positively correlated. Briefly, inconsistent variation families contain inconsistent items with different distributions of grouping attributes compared with other items, and vice versa.

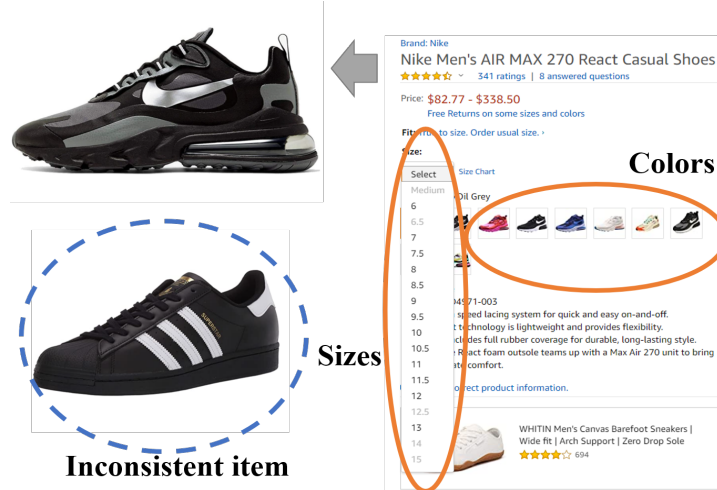


Figure 6.17: An example of correctly grouped variation family with variation theme attributes, and an inconsistent item.

we apply the pre-trained *Glove* embeddings [199] for text features of items.

The metric for this experiment is the PR-AUC for predicting inconsistent variation family (IVF) and consistent variation family (CVF), denoted as AUC-I and AUC-C respectively in Table 6.11. The results are reported based on the average of five runs.

6.3.2 Results on Datasets of Product Lines

As shown in Table 6.11, first we can see that by using the supervised pretext task (the last three rows), the model performances are consistently better than those without pre-trained pretext task in all product lines. Second, for both methods using pretext task, the pre-training strategy outperforms the joint training strategy by AUC-I metric and they are very close by AUC-C metric. The *Hyper-SAGNN* with the adapted pre-training stage also has relatively competitive performance compared with the proposed HyperGRL model. This indicates that in this real-world application of hyperedge classification, pre-training can indeed improve the base model as well as baseline method, and pre-training is generally a better training strategy than joint training. Third, by utilizing the adaptation stage with the node-level self-supervised pretext task (the last row), the model is able to further improve the AUC-I performance on Category-2 and Category-3 product lines, which demonstrates the effectiveness of the adaptation-aware pre-training of HyperGRL in this application scenario.

CHAPTER 7: CONCLUSION AND FUTURE DIRECTIONS

In this chapter, we summarize the major contributions of our research in this thesis, and then discuss the future directions in multi-network association.

7.1 SUMMARY OF THESIS WORK

In this thesis, we study the problem of multi-network association, and develop a variety of algorithms for resolving general pairwise and high-order multi-network association problems and their corresponding novel applications. According to our taxonomy of the problems and the technique types, four major contributions can be drawn from the thesis.

Task 1. Pairwise Association with Numerical Techniques. For the problem of pairwise association, on one hand, we have developed a family of Krylov subspace based algorithms (FASTEN [25]) to speed up and scale up the computation of Sylvester equation for graph mining. The key idea is to project the original equivalent linear system onto a Kronecker Krylov subspace. Based on that, we develop the following methods to further reduce complexity, including (1) implicitly representing the solution based on its low-rank structure, and (2) decomposing the original Sylvester equation into a set of small-scale, inter-correlated Sylvester equations. Two of the proposed algorithms have *linear* time and space complexity. Numerous experiments on real-world data show that the FASTEN family of algorithms (1) produce the exact solution, (2) are up to more than $10,000\times$ faster than the best known method, and (3) scale up to million-node graphs in about 100 seconds.

On the other hand, we study the interactive attributed subgraph matching problem and develop a family of efficient and effective algorithms (FIRST [17]) to address this problem according to different interactive scenarios. Specifically, we first recast the problem as a cross-network node similarity problem and show that the computation can be sped up by exploring the smoothness between initial and revised queries. We develop FIRST-Q and FIRST-N to handle the scenario where only node attribute is available, and FIRST-E to handle the scenario where both node and edge attribute are available. Numerous experiments on real world data show that our method lead up to $16\times$ speedup with more than 90% accuracy.

Task 2. Pairwise Association with Neural Techniques. First, we generalize the traditional Sylvester equation and develop Sylvester Multi-Graph Neural Network (SYMGNN [27]), which overcomes the limitations of the Sylvester equation. Second, we design a simplified GNN model with only user feature aggregation and interest propagation for social recommendation. Furthermore, we leverage both positive and negative samples for users'

preference diffusion between the representations of users and items in order to learn more compatible embeddings. Lastly, we propose a generative negative sampling approach to interpolate hard negative samples for improving model’s ability of generalization. Empirical results show that the proposed model significantly outperforms the state-of-the-art GNN-based models [28].

Task 3. High-order Association with Numerical Techniques. For the high-order association, we formulate the multi-way association inference problem as a convex optimization problem, and show that it can be solved optimally by a Sylvester tensor equation. We propose two fast algorithms (SYTE [15]) to solve this Sylvester tensor equation on both plain and attributed networks, with a *linear* complexity w.r.t. the size of input networks. Extensive empirical evaluations demonstrate (1) the effectiveness on a variety of multi-network mining tasks (e.g., multi-network alignment, multi-network node retrieval and high-order recommendation), and (2) the *linear* scalability of the proposed methods.

Task 4. High-order Association with Neural Techniques. We first study the multi-resolution multi-network embedding problem and develop efficient and effective algorithms (MRMINE and MRMINE+ [26]) to simultaneously learn the embeddings of multi-resolution multi-network objects in the same space. Specifically, we capture the context of such objects can be captured by the Cross-Resolution Cross-Network (CRCN) relation network. We then introduce a basic algorithm (MRMINE) to construct such network, and an accelerated algorithm (MRMINE+) which explores a more complex structure of the CRCN relation network (i.e. H-CRCN relation network) for reducing the time complexity. Numerous experiments on real-world data show that (1) our methods lead up to 19.71%, 28.20%, and 5.08% increase of accuracy in traditional network alignment, collective network alignment, and network classification, respectively; (2) our method can scale up to over 1M-node networks. Next, we study the problem of hypergraph representation learning, and propose an end-to-end hypergraph pre-training framework HyperGRL [29]. It incorporates bi-level self-supervised pretext tasks, and enables both transductive and inductive learning. HyperGRL does not require extra domain-specific datasets, and is adaptation-aware, which makes the pre-trained model more adaptive to downstream tasks compared to traditional pre-training methods. Extensive experiments demonstrate that: (1) HyperGRL shows significant improvements of downstream task performance and stability over baselines; (2) HyperGRL is efficient compared with traditional pre-training methods; (3) the proposed framework shows great applicability in real-world applications of online stores.

7.2 FUTURE DIRECTIONS

Multi-network mining is an active research field in data mining. Here, we provide several research directions which are beyond the scope of this thesis.

A - Active Multi-network Association/Embedding. Human-in-the-loop is an important aspect of Artificial Intelligence for robust machine learning/data mining model design. The motivation of active multi-network association/embedding is to let human interact with multi-network models, and the goal is to find the most informative node (set) for query in order to maximize the downstream task performance for the rest of the nodes. The first challenge is how to define and quantify node (set) information for query and the second challenge is how to identify such informative node (set). For active learning on multi-network embedding, previous work on applying active learning (AL) on network data, especially multi-network data is tightly coupled with traditional classification models such as Gaussian random fields, but the features of nodes are not used. AL models which utilize recent GNN architectures are still limited. Two possible directions on informativeness measure include a matching distribution-based certainty measurement [200] and an influence function based measurement [201] for the node (set) informativeness. Furthermore, two promising future directions for multi-network embedding with AL include combining AL with multi-network GNN models, and leveraging multi-armed bandit methods for active node selection strategies.

B - Adversarial Multi-network Association/Embedding. Existing adversarial attacks on network alignment are based on derivative-based importance score but very limited works exist on adversarial defense. On multi-networks, the complex data structure and the large data scale might further complicate the defense process compared to single network scenario. One future direction is to apply adversarial training for multi-network alignment/association tasks. For adversarial multi-network embedding, there are some pilot works on adversarial network embedding on single and simple networks (e.g., [202]). However, the multi-network structure complicates the embedding generation and discrimination. Another possible future direction is to combine the multi-network GNN model with the adversarial training to improve the robustness of embedding on multi-networks. Thanks to the generality of multi-network association and embedding, many applications could be benefited from the adversarial multi-network association/embedding technique, such as adversarial multi-network alignment, adversarial multi-network clustering/classification, etc.

C - Temporal Multi-network Association/Embedding. Real-world data is ever-changing overtime. Still, dynamic multi-network association/embedding-based methods are underexplored. A natural extension of multi-network mining algorithm is temporal multi-

network mining such as temporal multi-network association and embedding. However, direct application of static method onto temporal multi-network data is costly, since a static model needs to be re-trained at every time stamp. So, how to improve the efficiency without re-training the model from scratch is one major challenge. Another challenge is how to leverage the dynamics. For example, how to utilize the representation smoothness in dynamic multi-network data. For future directions, the first direction is to utilize matrix approximation, which is often used for dynamic algorithms to avoid unnecessary re-computation. The second direction is to leverage the mechanism of sequential models into discovering the dynamics of multi-network data, such as the Recurrent Neural Network (RNN)-based models [203], the Transformer [204], etc.

D - Hypergraphs for Recommender Systems. The multi-network data model, which models the high-order node relations, can also be used in the recommender system. For example, hypergraphs can be used for bundle/high-order recommendation. Here, the goal is to recommend a set of items to users (i.e., bundle recommendation [205]) or recommend items to a group of users (i.e., group recommendation [206]). Recently, hypergraphs are also adopted in the social recommendation task [78]. The major challenges of utilizing multi-network models in recommender systems are two-fold. First, how can we construct the hypergraphs from the data for item bundles/user sets in order to preserve the high-order node topology information? Second, how can we effectively model the high-order relation of hypergraphs in the recommendation task? Two potential directions are as follows. First, the problem could be simplified by dividing the process of hypergraph generation with the recommendation task, in which the hypergraph generation could often be conducted offline in an unsupervised style. Second, the items and users could be grouped together as hyperedges, and the group generator could be learned simultaneously with the recommendation task. Thus the model is also capable of generating new item bundles to a set of users.

E - Beyond Multi-network Association. Beyond the multi-network association problem, multi-network mining contains rich research topics. For instance, the topic of multi-network data model studies how to effectively represent complex real-world data with the assist of various multi-network data models for further mining. Representative multi-network data models include multi-view networks, multiplex networks, inter-dependent networks, and network of networks (NoN), etc. Multi-network mining also benefits numerous applications which use multi-network data. In the future, one interesting direction is to explore and create novel multi-network data models based on real-world application. Another direction is to improve the novel application performance by leveraging existing multi-network data models from the exact opposite direction.

REFERENCES

- [1] J. Tang, “Aminer: Toward understanding big scholar data,” in *Proceedings of the ninth ACM international conference on web search and data mining*, 2016, pp. 467–467.
- [2] A. Benton, R. Arora, and M. Dredze, “Learning multiview embeddings of twitter users,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2016, pp. 14–19.
- [3] T. Niu, S. Zhu, L. Pang, and A. El Saddik, “Sentiment analysis on multi-view social data,” in *International Conference on Multimedia Modeling*. Springer, 2016, pp. 15–27.
- [4] T. Yamada and P. Bork, “Evolution of biomolecular networks—lessons from metabolic and protein interactions,” *Nature Reviews Molecular Cell Biology*, vol. 10, no. 11, pp. 791–803, 2009.
- [5] T. Nepusz, H. Yu, and A. Paccanaro, “Detecting overlapping protein complexes in protein-protein interaction networks,” *Nature methods*, vol. 9, no. 5, pp. 471–472, 2012.
- [6] Y. Sun and J. Han, “Mining heterogeneous information networks: principles and methodologies,” *Synthesis Lectures on Data Mining and Knowledge Discovery*, vol. 3, no. 2, pp. 1–159, 2012.
- [7] C. Shi, Y. Li, J. Zhang, Y. Sun, and S. Y. Philip, “A survey of heterogeneous information network analysis,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 1, pp. 17–37, 2016.
- [8] G. L. Ciampaglia, P. Shiralkar, L. M. Rocha, J. Bollen, F. Menczer, and A. Flammini, “Computational fact checking from knowledge networks,” *PloS one*, vol. 10, no. 6, p. e0128193, 2015.
- [9] X. V. Lin, R. Socher, and C. Xiong, “Multi-hop knowledge graph reasoning with reward shaping,” *arXiv preprint arXiv:1808.10568*, 2018.
- [10] X. Huang, J. Zhang, D. Li, and P. Li, “Knowledge graph embedding based question answering,” in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 105–113.
- [11] G. Lee, J. Ko, and K. Shin, “Hypergraph motifs: Concepts, algorithms, and discoveries,” *arXiv preprint arXiv:2003.01853*, 2020.
- [12] L. Sun, S. Ji, and J. Ye, “Hypergraph spectral learning for multi-label classification,” in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 668–676.

- [13] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, “Hyper-gcn: A new method of training graph convolutional networks on hypergraphs,” *arXiv preprint arXiv:1809.02589*, 2018.
- [14] M. Zhang, Z. Cui, S. Jiang, and Y. Chen, “Beyond link prediction: Predicting hyperlinks in adjacency space,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
- [15] B. Du, L. Liu, and H. Tong, “Sylvester tensor equation for multi-way association,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 311–321.
- [16] Z. Li, W. Zhang, R. S. Huang, and R. Kuang, “Learning a low-rank tensor of pharmacogenomic multi-relations from biomedical networks,” in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 409–418.
- [17] B. Du, S. Zhang, N. Cao, and H. Tong, “First: Fast interactive attributed subgraph matching,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1447–1456.
- [18] X. Chu, X. Fan, D. Yao, Z. Zhu, J. Huang, and J. Bi, “Cross-network embedding for multi-network alignment,” in *The world wide web conference*, 2019, pp. 273–284.
- [19] C.-S. Liao, K. Lu, M. Baym, R. Singh, and B. Berger, “Isorankn: spectral methods for global alignment of multiple protein networks,” *Bioinformatics*, vol. 25, no. 12, pp. i253–i258, 2009.
- [20] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, “struc2vec: Learning node representations from structural identity,” in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 385–394.
- [21] J. Ni, 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*, 2014, pp. 1356–1365.
- [22] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, “Structural deep embedding for hyper-networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
- [23] B. Bahmani, A. Chowdhury, and A. Goel, “Fast incremental and personalized pagerank,” *arXiv preprint arXiv:1006.2880*, 2010.
- [24] R. Singh, J. Xu, and B. Berger, “Pairwise global alignment of protein interaction networks by matching neighborhood topology,” in *Annual International Conference on Research in Computational Molecular Biology*. Springer, 2007, pp. 16–31.
- [25] B. Du and H. Tong, “Fasten: Fast sylvester equation solver for graph mining,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1339–1347.

- [26] B. Du and H. Tong, “Mrmine: Multi-resolution multi-network embedding,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 479–488.
- [27] B. Du and H. Tong, “Geometric matrix completion via sylvester multi-graph neural network,” 2021.
- [28] B. Du and H. Tong, “Neural multi-network diffusion for social recommendation,” 2021.
- [29] B. Du, C. Yuan, R. Barton, T. Neiman, and H. Tong, “Hypergraph pre-training with graph neural networks,” *arXiv preprint arXiv:2105.10862*, 2021.
- [30] S. Zhang and H. Tong, “Final: Fast attributed network alignment,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1345–1354.
- [31] J. Ni, S. Chang, X. Liu, W. Cheng, H. Chen, D. Xu, and X. Zhang, “Co-regularized deep multi-network embedding,” in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 469–478.
- [32] Y. Yan, D. Luo, J. Ni, H. Fei, W. Fan, X. Yu, J. Yen, and X. Zhang, “Local graph clustering by multi-network random walk with restart,” in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2018, pp. 490–501.
- [33] D. Luo, J. Ni, S. Wang, Y. Bian, X. Yu, and X. Zhang, “Deep multi-graph clustering via attentive cross-graph association,” in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 393–401.
- [34] F. Monti, M. M. Bronstein, and X. Bresson, “Geometric matrix completion with recurrent multi-graph neural networks,” *arXiv preprint arXiv:1704.06803*, 2017.
- [35] A. Mongia and A. Majumdar, “Matrix completion on multiple graphs: Application in collaborative filtering,” *Signal Processing*, vol. 165, pp. 144–148, 2019.
- [36] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst, “Matrix completion on graphs,” *arXiv preprint arXiv:1408.1717*, 2014.
- [37] X. Kong, J. Zhang, and P. S. Yu, “Inferring anchor links across multiple heterogeneous social networks,” in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 179–188.
- [38] R. Liu, W. Cheng, H. Tong, W. Wang, and X. Zhang, “Robust multi-network clustering via joint cross-domain cluster alignment,” in *2015 IEEE International Conference on Data Mining*. IEEE, 2015, pp. 291–300.
- [39] L. Liu, W. K. Cheung, X. Li, and L. Liao, “Aligning users across social networks using network embedding,” in *Ijcai*, 2016, pp. 1774–1780.

- [40] B. Yan and C. Wang, “Graphae: Adaptive embedding across graphs,” in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1958–1961.
- [41] M. Zhang and Y. Chen, “Inductive matrix completion based on graph neural networks,” *arXiv preprint arXiv:1904.12058*, 2019.
- [42] R. v. d. Berg, T. N. Kipf, and M. Welling, “Graph convolutional matrix completion,” *arXiv preprint arXiv:1706.02263*, 2017.
- [43] D. M. Nguyen, E. Tsiligianni, and N. Deligiannis, “Extendable neural matrix completion,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 6328–6332.
- [44] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, “Graph neural networks for social recommendation,” in *The World Wide Web Conference*, 2019, pp. 417–426.
- [45] J. Li, S. Zhang, T. Liu, C. Ning, Z. Zhang, and W. Zhou, “Neural inductive matrix completion with graph convolutional networks for mirna-disease association prediction,” *Bioinformatics*, vol. 36, no. 8, pp. 2538–2546, 2020.
- [46] L. Ou-Yang, H. Yan, and X.-F. Zhang, “A multi-network clustering method for detecting protein complexes from multiple heterogeneous networks,” *BMC bioinformatics*, vol. 18, no. 13, pp. 23–34, 2017.
- [47] I. Makarov, D. Kiselev, N. Nikitinsky, L. Subelj, O. M. Elzeki, M. Shams, S. Sarhan, M. Abd Elfattah, A. E. Hassanien, H. Salem et al., “Survey on graph embeddings and their applications to machine learning problems on graphs,” *PeerJ Computer Science*, 2021.
- [48] F. Zhou, C. Cao, G. Trajcevski, K. Zhang, T. Zhong, and J. Geng, “Fast network alignment via graph meta-learning,” in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 686–695.
- [49] L. Xu, X. Wei, J. Cao, and P. S. Yu, “Embedding of embedding (eoe) joint embedding for coupled heterogeneous networks,” in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017, pp. 741–749.
- [50] M. Radmanesh, A. A. Rezaei, N. Al Khafaf, and M. Jalili, “Topological deep network embedding,” in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC)*. IEEE, 2020, pp. 476–481.
- [51] H. Xue, J. Peng, J. Li, and X. Shang, “Integrating multi-network topology via deep semi-supervised node embedding,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 2117–2120.
- [52] M. Milano, P. H. Guzzi, and M. Cannataro, “Using multi network alignment for analysis of connectomes,” *Procedia Computer Science*, vol. 108, pp. 1155–1164, 2017.

- [53] D. Cai, X. He, J. Han, and T. S. Huang, “Graph regularized nonnegative matrix factorization for data representation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 8, pp. 1548–1560, 2010.
- [54] A. Narita, K. Hayashi, R. Tomioka, and H. Kashima, “Tensor factorization using auxiliary information,” *Data Mining and Knowledge Discovery*, vol. 25, no. 2, pp. 298–324, 2012.
- [55] M. Nickel, V. Tresp, and H.-P. Kriegel, “A three-way model for collective learning on multi-relational data,” in *Icml*, 2011.
- [56] H. Liu and Y. Yang, “Bipartite edge prediction via transductive learning over product graphs,” in *International Conference on Machine Learning*. PMLR, 2015, pp. 1880–1888.
- [57] H. Liu and Y. Yang, “Cross-graph learning of multi-relational associations,” in *International Conference on Machine Learning*. PMLR, 2016, pp. 2235–2243.
- [58] Z. Li, R. Petegrosso, S. Smith, D. Sterling, G. Karypis, and R. Kuang, “Scalable label propagation for multi-relational learning on tensor product graph,” *arXiv preprint arXiv:1802.07379*, 2018.
- [59] D. Zhou, J. Huang, and B. Schölkopf, “Learning with hypergraphs: Clustering, classification, and embedding,” *Advances in neural information processing systems*, vol. 19, pp. 1601–1608, 2006.
- [60] D. Li, Z. Xu, S. Li, and X. Sun, “Link prediction in social networks based on hypergraph,” in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, pp. 41–42.
- [61] Q. Liu, Y. Huang, and D. N. Metaxas, “Hypergraph with sampling for image retrieval,” *Pattern Recognition*, vol. 44, no. 10-11, pp. 2255–2262, 2011.
- [62] S. Tan, J. Bu, C. Chen, B. Xu, C. Wang, and X. He, “Using rich social media information for music recommendation via hypergraph model,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 7, no. 1, pp. 1–22, 2011.
- [63] S. Tan, Z. Guan, D. Cai, X. Qin, J. Bu, and C. Chen, “Mapping users across networks by manifold alignment on hypergraph,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014.
- [64] T. Hwang, Z. Tian, R. Kuangy, and J.-P. Kocher, “Learning on weighted hypergraphs to integrate protein interactions and gene expressions for cancer outcome prediction,” in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 293–302.

- [65] Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher, “Metafac: community discovery via relational hypergraph factorization,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 527–536.
- [66] S.-e. Yoon, H. Song, K. Shin, and Y. Yi, “How much and when do we need higher-order information in hypergraphs? a case study on hyperedge prediction,” in *Proceedings of The Web Conference 2020*, 2020, pp. 2627–2633.
- [67] A. Sharma, J. Srivastava, and A. Chandra, “Predicting multi-actor collaborations using hypergraphs,” *arXiv preprint arXiv:1401.6404*, 2014.
- [68] H. Xue, J. Peng, and X. Shang, “Deep feature learning of multi-network topology for node classification,” *arXiv preprint arXiv:1809.02394*, 2018.
- [69] R. Trivedi, B. Sisman, J. Ma, C. Faloutsos, H. Zha, and X. L. Dong, “Linknbed: Multi-graph representation learning with entity linkage,” *arXiv preprint arXiv:1807.08447*, 2018.
- [70] Y. Liu, C. Liang, X. He, J. Peng, Z. Zheng, and J. Tang, “Modelling high-order social relations for item recommendation,” *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [71] M. Jiang, P. Cui, F. Wang, Q. Yang, W. Zhu, and S. Yang, “Social recommendation across multiple relational domains,” in *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 1422–1431.
- [72] D. Yang, B. Qu, J. Yang, and P. Cudre-Mauroux, “Revisiting user mobility and social relationships in lbsns: A hypergraph embedding approach,” in *The world wide web conference*, 2019, pp. 2147–2157.
- [73] D. Yang, B. Qu, J. Yang, and P. Cudré-Mauroux, “Lbsn2vec++: Heterogeneous hypergraph embedding for location-based social networks,” *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [74] J. Shang, S. Huang, D. Zhang, Z. Peng, D. Liu, Y. Li, and L. Xu, “Rne2vec: information diffusion popularity prediction based on repost network embedding,” *Computing*, vol. 103, no. 2, pp. 271–289, 2021.
- [75] D. Arya and M. Worring, “Exploiting relational information in social networks using geometric deep learning on hypergraphs,” in *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, 2018, pp. 117–125.
- [76] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, “Hypergraph neural networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3558–3565.
- [77] R. Zhang, Y. Zou, and J. Ma, “Hyper-saggnn: a self-attention based graph neural network for hypergraphs,” *arXiv preprint arXiv:1911.02613*, 2019.

- [78] J. Yu, H. Yin, J. Li, Q. Wang, N. Q. V. Hung, and X. Zhang, “Self-supervised multi-channel hypergraph convolutional network for social recommendation,” *arXiv preprint arXiv:2101.06448*, 2021.
- [79] J. Jiang, Y. Wei, Y. Feng, J. Cao, and Y. Gao, “Dynamic hypergraph neural networks.” in *IJCAI*, 2019, pp. 2635–2641.
- [80] N. Yadati, V. Nitin, M. Nimishakavi, P. Yadav, A. Louis, and P. Talukdar, “Link prediction in hypergraphs using graph convolutional networks,” 2018.
- [81] D. Koutra, H. Tong, and D. Lubensky, “Big-align: Fast bipartite graph alignment,” in *2013 IEEE 13th International Conference on Data Mining*. IEEE, 2013, pp. 389–398.
- [82] M. Bayati, D. F. Gleich, A. Saberi, and Y. Wang, “Message-passing algorithms for sparse network alignment,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 7, no. 1, pp. 1–31, 2013.
- [83] J. Zhang and S. Y. Philip, “Multiple anonymized social networks alignment,” in *2015 IEEE International Conference on Data Mining*. IEEE, 2015, pp. 599–608.
- [84] S. Mohammadi, D. F. Gleich, T. G. Kolda, and A. Grama, “Triangular alignment (tame): A tensor-based approach for higher-order network alignment,” *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 14, no. 6, pp. 1446–1458, 2016.
- [85] S. Zhang, H. Tong, R. Maciejewski, and T. Eliassi-Rad, “Multilevel network alignment,” in *The World Wide Web Conference*, 2019, pp. 2344–2354.
- [86] R. Kondor, N. Teneva, and V. Garg, “Multiresolution matrix factorization,” in *International Conference on Machine Learning*. PMLR, 2014, pp. 1620–1628.
- [87] S. Zhang and H. Tong, “Network alignment: Recent advances and future directions,” in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3521–3522.
- [88] M. Jamali and M. Ester, “A matrix factorization technique with trust propagation for recommendation in social networks,” in *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 135–142.
- [89] G. Guo, J. Zhang, and N. Yorke-Smith, “Trustsvd: Collaborative filtering with both the explicit and implicit influence of user trust and of item ratings,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.
- [90] M. Jiang, P. Cui, F. Wang, W. Zhu, and S. Yang, “Scalable recommendation with social contextual information,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 11, pp. 2789–2802, 2014.
- [91] L. Wu, P. Sun, R. Hong, Y. Ge, and M. Wang, “Collaborative neural social recommendation,” *IEEE transactions on systems, man, and cybernetics: systems*, 2018.

- [92] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph convolutional neural networks for web-scale recommender systems,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.
- [93] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, “Neural graph collaborative filtering,” in *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, 2019, pp. 165–174.
- [94] L. Wu, P. Sun, Y. Fu, R. Hong, X. Wang, and M. Wang, “A neural influence diffusion model for social recommendation,” in *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, 2019, pp. 235–244.
- [95] L. Wu, J. Li, P. Sun, R. Hong, Y. Ge, and M. Wang, “Diffnet++: A neural influence and interest diffusion network for social recommendation,” *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [96] C. Song, B. Wang, Q. Jiang, Y. Zhang, R. He, and Y. Hou, “Social recommendation with implicit social influence,” in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021, pp. 1788–1792.
- [97] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, “Lightgcn: Simplifying and powering graph convolution network for recommendation,” in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020, pp. 639–648.
- [98] J. Yu, H. Yin, J. Li, M. Gao, Z. Huang, and L. Cui, “Enhance social recommendation with adversarial graph convolutional networks,” *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [99] G. Jeh and J. Widom, “Simrank: a measure of structural-context similarity,” in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 538–543.
- [100] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, “Structural deep embedding for hyper-networks,” *arXiv preprint arXiv:1711.10146*, 2017.
- [101] J. Payne, “Deep hyperedges: a framework for transductive and inductive learning on hypergraphs,” *arXiv preprint arXiv:1910.02633*, 2019.
- [102] D. G. Corneil and C. C. Gotlieb, “An efficient algorithm for graph isomorphism,” *J. ACM*, vol. 17, no. 1, pp. 51–64, Jan. 1970. [Online]. Available: <http://doi.acm.org/10.1145/321556.321562>

- [103] W.-S. Han, J. Lee, and J.-H. Lee, “Turboiso: Towards ultrafast and robust subgraph isomorphism search in large graph databases,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’13. New York, NY, USA: ACM, 2013. [Online]. Available: <http://doi.acm.org/10.1145/2463676.2465300> pp. 337–348.
- [104] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, “Efficient subgraph matching by postponing cartesian products,” in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD ’16. New York, NY, USA: ACM, 2016. [Online]. Available: <http://doi.acm.org/10.1145/2882903.2915236> pp. 1199–1214.
- [105] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, “Fast best-effort pattern matching in large attributed graphs,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 737–746.
- [106] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu, “Automatic multimedia cross-modal correlation discovery,” in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 653–658.
- [107] H. Tong, C. Faloutsos, and J.-Y. Pan, “Fast random walk with restart and its applications,” 2006.
- [108] H. Tong and C. Faloutsos, “Center-piece subgraphs: problem definition and fast solutions,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 404–413.
- [109] Y. Tian and J. M. Patel, “Tale: A tool for approximate large graph matching,” in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 963–972.
- [110] M. Mongiovi, R. Di Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan, “Sigma: a set-cover-based inexact graph matching algorithm,” *Journal of bioinformatics and computational biology*, vol. 8, no. 02, pp. 199–218, 2010.
- [111] S. B. Roy, T. Eliassi-Rad, and S. Papadimitriou, “Fast best-effort search on graphs with multiple attributes,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 755–768, 2015.
- [112] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan, “Nema: Fast graph search with label similarity,” in *Proceedings of the VLDB Endowment*, vol. 6, no. 3. VLDB Endowment, 2013, pp. 181–192.
- [113] L. Liu, B. Du, H. Tong et al., “G-finder: Approximate attributed subgraph matching,” in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 513–522.

- [114] M. Heimann, H. Shen, T. Safavi, and D. Koutra, “Regal: Representation learning-based graph alignment,” in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 117–126.
- [115] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.
- [116] Y. Zhang, “Browser-oriented universal cross-site recommendation and explanation based on user browsing logs,” in *Proceedings of the 8th ACM Conference on Recommender systems*, 2014.
- [117] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, “Graph kernels,” *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.
- [118] K. Onuma, H. Tong, and C. Faloutsos, “Tangent: a novel, ‘surprise me’, recommendation algorithm,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.
- [119] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu, “Fast computation of simrank for static and dynamic information networks,” in *Proceedings of the 13th International Conference on Extending Database Technology*, 2010, pp. 465–476.
- [120] S. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph, “Fast computation of graph kernels,” in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 2006.
- [121] U. Kang, H. Tong, and J. Sun, “Fast random walk graph kernel,” in *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012.
- [122] L. Li, H. Tong, Y. Xiao, and W. Fan, “Cheetah: fast graph kernel tracking on dynamic graphs,” in *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 2015.
- [123] A. El Guennoui, K. Jbilou, and A. Riquet, “Block krylov subspace methods for solving large sylvester equations,” *Numerical Algorithms*, vol. 29, no. 1, pp. 75–96, 2002.
- [124] D. Y. Hu and L. Reichel, “Krylov-subspace methods for the sylvester equation,” *Linear Algebra and its Applications*, vol. 172, pp. 283–313, 1992.
- [125] S. Zhang, H. Tong, J. Tang, J. Xu, and W. Fan, “ineat: Incomplete network alignment,” in *2017 IEEE International Conference on Data Mining (ICDM)*, 2017.
- [126] P. Tseng, “Convergence of a block coordinate descent method for nondifferentiable minimization,” *Journal of optimization theory and applications*, 2001.
- [127] J. Tang, 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*, 2008.

- [128] A. Prado, M. Plantevit, C. Robardet, and J.-F. Boulicaut, “Mining graph topological patterns: Finding covariations among vertex descriptors,” *Knowledge and Data Engineering, IEEE Transactions on*, 2013.
- [129] Y. Zhang, J. Tang, Z. Yang, J. Pei, and P. S. Yu, “Cosnet: Connecting heterogeneous social networks with local and global consistency,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.
- [130] R. Pienta, A. Tamersoy, H. Tong, and D. H. Chau, “Mage: Matching approximate patterns in richly-attributed graphs,” in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 585–590.
- [131] N. Cao, Y.-R. Lin, L. Li, and H. Tong, “g-miner: Interactive visual group mining on multivariate graphs,” in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 279–288.
- [132] L. Li, 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*. ACM, 2015, pp. 636–646.
- [133] L. Li, H. Tong, N. Cao, K. Ehrlich, Y.-R. Lin, and N. Buchler, “Teamopt: Interactive team optimization in big networks,” in *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*. ACM, 2016, pp. 2485–2487.
- [134] W. W. Piegorsch and G. Casella, “Erratum: inverting a sum of matrices,” *SIAM review*, vol. 32, no. 3, p. 470, 1990.
- [135] K. B. Petersen, M. S. Pedersen et al., “The matrix cookbook,” *Technical University of Denmark*, vol. 7, p. 15, 2008.
- [136] G. Golub, S. Nash, and C. Van Loan, “A hessenberg-schur method for the problem $ax + xb = c$,” *IEEE Transactions on Automatic Control*, vol. 24, no. 6, pp. 909–913, 1979.
- [137] E. L. Wachspress, “Iterative solution of the lyapunov matrix equation,” *Applied Mathematics Letters*, vol. 1, no. 1, pp. 87–90, 1988.
- [138] P. Benner, “Factorized solution of sylvester equations with applications in control,” *sign (H)*, 2004.
- [139] A. Agovic, A. Banerjee, and S. Chatterjee, “Probabilistic matrix addition,” in *ICML*, 2011.
- [140] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [141] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph attention networks,” *arXiv preprint arXiv:1710.10903*, 2017.

- [142] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” *arXiv preprint arXiv:1810.00826*, 2018.
- [143] K.-L. Yao, W.-J. Li, J. Yang, and X. Lu, “Convolutional geometric matrix completion,” *arXiv preprint arXiv:1803.00754*, 2018.
- [144] T. Yoshida, I. Takeuchi, and M. Karasuyama, “Distance metric learning for graph structured data,” *Machine Learning*, pp. 1–47, 2021.
- [145] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. El Ghaoui, “Implicit graph neural networks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 984–11 995, 2020.
- [146] F. Wang and J. Sun, “Survey on distance metric learning and dimensionality reduction in data mining,” *Data mining and knowledge discovery*, vol. 29, no. 2, pp. 534–564, 2015.
- [147] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Y. Tsai, “Implicit deep learning,” *arXiv preprint arXiv:1908.06315*, vol. 2, 2019.
- [148] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *arXiv preprint arXiv:1706.02216*, 2017.
- [149] M. Zhang and Y. Chen, “Link prediction based on graph neural networks,” *Advances in Neural Information Processing Systems*, vol. 31, pp. 5165–5175, 2018.
- [150] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, “Graph matching networks for learning the similarity of graph structured objects,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 3835–3845.
- [151] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, “Recommender systems with social regularization,” in *Proceedings of the fourth ACM international conference on Web search and data mining*, 2011, pp. 287–296.
- [152] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.
- [153] Y. Koren, “Factorization meets the neighborhood: a multifaceted collaborative filtering model,” in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 426–434.
- [154] A. Mnih and R. R. Salakhutdinov, “Probabilistic matrix factorization,” *Advances in neural information processing systems*, vol. 20, 2007.
- [155] H. Yu, H. Wang, and J. Wu, “Mixup without hesitation,” *arXiv preprint arXiv:2101.04342*, 2021.

- [156] T. Huang, Y. Dong, M. Ding, Z. Yang, W. Feng, X. Wang, and J. Tang, “Mixgcf: An improved training method for graph neural network-based recommender systems,” 2021.
- [157] X. Li, L. Wen, Y. Deng, F. Feng, X. Hu, L. Wang, and Z. Fan, “Graph neural network with curriculum learning for imbalanced node classification,” *arXiv preprint arXiv:2202.02529*, 2022.
- [158] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, “Curgraph: Curriculum learning for graph classification,” in *Proceedings of the Web Conference 2021*, 2021, pp. 1238–1248.
- [159] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, “Bpr: Bayesian personalized ranking from implicit feedback,” *arXiv preprint arXiv:1205.2618*, 2012.
- [160] C. Chen, 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*, 2016, pp. 765–774.
- [161] X. Zhu, Z. Ghahramani, and J. D. Lafferty, “Semi-supervised learning using gaussian fields and harmonic functions,” in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.
- [162] Z. Liang, M. Xu, M. Teng, and L. Niu, “Netalign: a web-based tool for comparison of protein interaction networks,” *Bioinformatics*, vol. 22, no. 17, pp. 2175–2177, 2006.
- [163] R. Singh, J. Xu, and B. Berger, “Global alignment of multiple protein interaction networks with application to functional orthology detection,” *Proceedings of the National Academy of Sciences*, vol. 105, no. 35, pp. 12 763–12 768, 2008.
- [164] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.
- [165] D. Kressner and C. Tobler, “Krylov subspace methods for linear systems with tensor product structure,” *SIAM journal on matrix analysis and applications*, vol. 31, no. 4, pp. 1688–1714, 2010.
- [166] Q. Zhou, L. Li, N. Cao, L. Ying, and H. Tong, “Admiring: Adversarial multi-network mining,” in *ICDM*, 2019.
- [167] A. Prado, M. Plantevit, C. Robardet, and J.-F. Boulicaut, “Mining graph topological patterns: Finding covariations among vertex descriptors,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 9, pp. 2090–2104, 2012.
- [168] I. Cantador, P. Brusilovsky, and T. Kuflik, “2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011).”

- [169] J. Zhang and P. S. Yu, “Integrated anchor and social link predictions across social networks,” in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [170] X. Wu, B. Shi, Y. Dong, C. Huang, and N. Chawla, “Neural tensor factorization,” *arXiv preprint arXiv:1802.04416*, 2018.
- [171] Y. Yao, H. Tong, G. Yan, F. Xu, X. Zhang, B. K. Szymanski, and J. Lu, “Dual-regularized one-class collaborative filtering,” in *CIKM 2014, Shanghai, China, November 3-7, 2014*, 2014, pp. 759–768.
- [172] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.
- [173] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.
- [174] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “Line: Large-scale information network embedding,” in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.
- [175] J. Li, C. Chen, H. Tong, and H. Liu, “Multi-layered network embedding,” in *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 2018, pp. 684–692.
- [176] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, “sub-graph2vec: Learning distributed representations of rooted sub-graphs from large graphs,” 2016.
- [177] P. Yanardag and S. Vishwanathan, “Deep graph kernels,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.
- [178] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, “graph2vec: Learning distributed representations of graphs,” *13th International Workshop on Mining and Learning with Graphs*, 2017.
- [179] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler-lehman graph kernels.” *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.
- [180] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

- [181] E. Zhong, W. Fan, J. Wang, L. Xiao, and Y. Li, “Comsoc: adaptive transfer of user behaviors over composite social network,” in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 696–704.
- [182] M. Vaida and K. Purcell, “Hypergraph link prediction: Learning drug interaction networks embeddings,” in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 1860–1865.
- [183] S. H. Sindrup and T. S. Jensen, “Efficacy of pharmacological treatments of neuropathic pain: an update and effect related to mechanism of drug action,” *PAIN®*, vol. 83, no. 3, pp. 389–400, 1999.
- [184] J. Piñero, J. M. Ramírez-Anguita, J. Saüch-Pitarch, F. Ronzano, E. Centeno, F. Sanz, and L. I. Furlong, “The disgenet knowledge platform for disease genomics: 2019 update,” *Nucleic acids research*, vol. 48, no. D1, pp. D845–D855, 2020.
- [185] J. Motl and O. Schulte, “The ctu prague relational learning repository,” *arXiv preprint arXiv:1511.03086*, 2015.
- [186] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, “Hypergen: A new method for training graph convolutional networks on hypergraphs,” in *Advances in Neural Information Processing Systems*, 2019, pp. 1511–1522.
- [187] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [188] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, “Strategies for pre-training graph neural networks,” *arXiv preprint arXiv:1905.12265*, 2019.
- [189] W. Jin, T. Derr, H. Liu, Y. Wang, S. Wang, Z. Liu, and J. Tang, “Self-supervised learning on graphs: Deep insights and new direction,” *arXiv preprint arXiv:2006.10141*, 2020.
- [190] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, “Gpt-gnn: Generative pre-training of graph neural networks,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1857–1867.
- [191] Y. Lu, X. Jiang, Y. Fang, and C. Shi, “Learning to pre-train graph neural networks,” 2021.
- [192] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” *arXiv preprint arXiv:1704.01212*, 2017.
- [193] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

- [194] O. Vinyals, S. Bengio, and M. Kudlur, “Order matters: Sequence to sequence for sets,” *arXiv preprint arXiv:1511.06391*, 2015.
- [195] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. R. Salakhutdinov, and A. J. Smola, “Deep sets,” in *Advances in neural information processing systems*, 2017, pp. 3391–3401.
- [196] L. Liu, T. Zhou, G. Long, J. Jiang, and C. Zhang, “Learning to propagate for graph meta-learning,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [197] D. Mandal, S. Medya, B. Uzzi, and C. Aggarwal, “Metalearning with graph neural networks: Methods and applications,” *ACM SIGKDD Explorations Newsletter*, vol. 23, no. 2, pp. 13–22, 2022.
- [198] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU, “Query-driven active surveying for collective classification,” in *10th International Workshop on Mining and Learning with Graphs*, vol. 8, 2012.
- [199] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
- [200] E. Malmi, A. Gionis, and E. Terzi, “Active network alignment: a matching-based approach,” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 1687–1696.
- [201] Q. Zhou, L. Li, X. Wu, N. Cao, L. Ying, and H. Tong, “Attent: Active attributed network alignment,” in *Proceedings of the Web Conference 2021*, 2021, pp. 3896–3906.
- [202] Q. Dai, Q. Li, J. Tang, and D. Wang, “Adversarial network embedding,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
- [203] M. Sundermeyer, R. Schlüter, and H. Ney, “Lstm neural networks for language modeling,” in *Thirteenth annual conference of the international speech communication association*, 2012.
- [204] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.
- [205] J. Chang, C. Gao, X. He, D. Jin, and Y. Li, “Bundle recommendation and generation with graph neural networks,” *IEEE Transactions on Knowledge and Data Engineering*, 2021.
- [206] D. Cao, X. He, L. Miao, Y. An, C. Yang, and R. Hong, “Attentive group recommendation,” in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 645–654.